Introduction to Industrial Informatics and System Engineering in Factory Automation

Final iteration

Tampere University of Technology

Group 9

Seyedsina Miri

Rongwei Ma

Enbo Chen

Content

Introduction

In this report, it is can divided into two section. The first section is mainly concern with UML files. In this report, we are totally includes the activity diagram, class diagram, sequence diagram, state diagram, use case diagram, project libre, cost benefit analysis and crud matrix. The second section is node.js project, we are focusing on how to build the server and client that can achieve monitoring and operating synchronize.

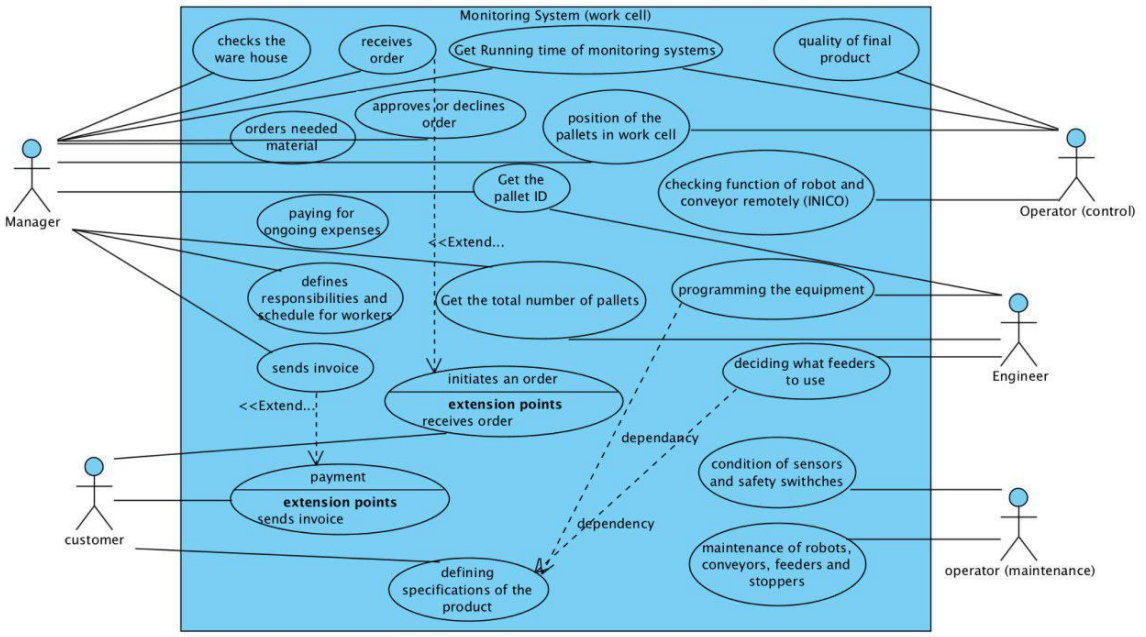
First iteration with modified diagrams

For the first iteration of assignment, we developed a case scenario for our project. For our case scenario which is the process of monitoring the work cell we limit our system vision to a work cell in the factory floor and decide what has to be managed in our project. First we divide the task of monitoring into following main sub-categories (project management, business model, environment, requirements and design) then we decide how the project can be managed.

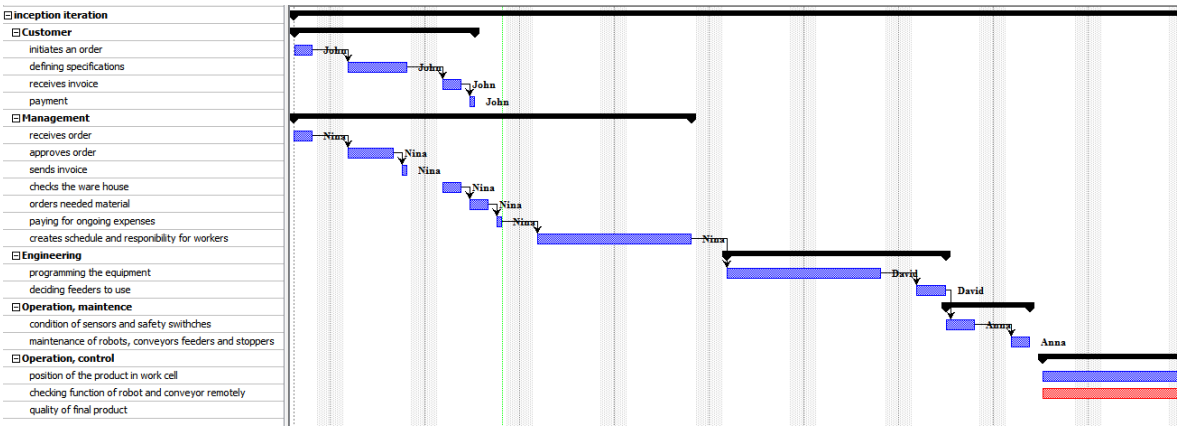
In project of monitoring the work cell we need to gather following data:

- Who has the authority to run the system and when this can happen? (correlation of manager and the customers)
- What are specifications of the product (mounted on a pallet) entering the work cell and what has to be done on it? (Is there any process that needs to be done on the pallet or it just bypasses the work cell?)
- What is the position of our pallet at each moment in our work cell?
- What is the status of our equipment (robot, conveyor, feeders, sensors and stoppers)
- How the safety of the system is maintained?
- What are engineers' responsibilities for programming equipment in the work cell?
- What are the schedules for maintenance of our equipment in the work cell?
- Which operator has the authority for maintenance of the system?
- How all these data can be collected into our database and analyzed for monitoring?

In order to start visualizing the task of monitoring we start with creating a use case diagram. Use case diagrams are used to gather the requirements of a system including internal and external influences. These requirements are mostly design requirements. So when a system is analyzed to gather its functionalities, use cases are prepared and actors are identified. For our monitoring system the use case diagram that we have prepared is followed:



Also, for the first iteration we developed a WBS. In our WBS, we defined all the tasks (monitoring and so) according to our use case diagram and considered the recourses and the schedule for the whole task in details. A screen shot of our WBS is presented below:



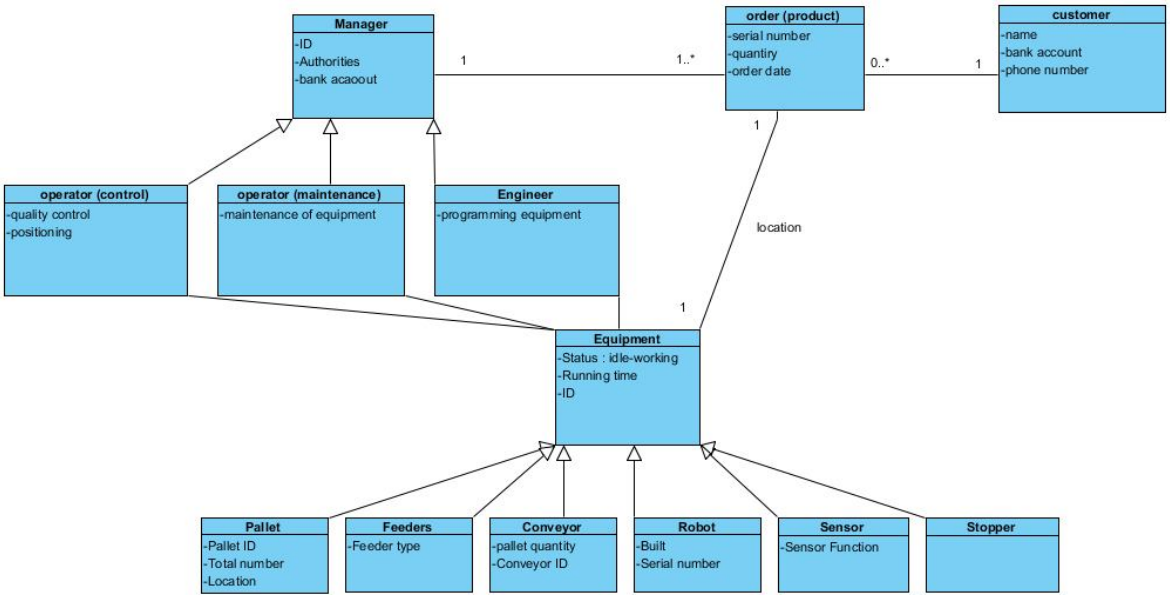
And for the last part of the assignment which is cost-benefit evaluation, we imagined the following costs and benefits for our project and put them all in a spread sheet:

		Current Year (CY)	CY +1	CY +2	CY +3	CY +4	CY +5			
Costs									**Cost Benefit Analysis**	
1	Purchase	$ 200,000.00	$ 200.00	$ 100.00	$ 50.00	$ 50.00	$ 50.00		Total PV Benefits	$ 1,156,873.57
2	Labor	$ 20,000.00	$ 22,000.00	$ 23,000.00	$ 23,500.00	$ 24,000.00	$ 25,000.00		Total PV Costs	$ 381,205.10
3	Maintenance	$ 1,000.00	$ 1,000.00	$ 1,050.00	$ 1,100.00	$ 1,150.00	$ 1,200.00		**NET BENEFIT**	**775,668.48**
4	Energy	$ 5,000.00	$ 5,000.00	$ 5,000.00	$ 5,000.00	$ 5,000.00	$ 5,000.00			
5	Software	$ 10,000.00	$ 1,500.00	$ 1,200.00	$ 1,000.00	$ 1,000.00	$ 1,000.00			
	Total Costs (Future Value)	$ 236,000.00	$ 29,700.00	$ 30,350.00	$ 30,650.00	$ 31,200.00	$ 32,250.00			
	Total Costs (Present Value)	**$ 236,000.00**	**$ 29,117.65**	**$ 29,171.47**	**$ 28,882.18**	**$ 28,823.98**	**$ 29,209.82**	**$ 381,205.10**		
Benefits										
1	Sales of Nokia Phones X10	$ 20,000.00	$ 40,000.00	$ 75,000.00	$ 80,000.00	$ 100,000.00	$ 150,000.00			
2	Sales of Nokia Phones X100	$ 23,000.00	$ 20,000.00	$ 32,000.00	$ 23,100.00	$ 222,000.00	$ 32,320.00			
3	Sales of Nokia Phones X1000	$ 40,000.00	$ 50,000.00	$ 65,200.00	$ 76,000.00	$ 87,000.00	$ 98,000.00			
	Total Benefits (Future Value)	$ 83,000.00	$ 110,000.00	$ 172,200.00	$ 179,100.00	$ 409,000.00	$ 280,320.00			
	Total Benefits (Present Value)	**$ 83,000.00**	**$ 107,843.14**	**$ 165,513.26**	**$ 168,769.93**	**$ 377,852.78**	**$ 253,894.46**	**$ 1,156,873.57**		

Our cost-benefit evaluation might not seem to be so realistic and the reason for that is because the scope of our project is not defined so clearly. So these are just the numbers for an imaginary case.

Second iteration with modified diagrams

For the second iteration, we created class diagram for our monitoring system. The class diagram describes the attributes and operations of a class and also the constraints imposed on the system. For our system, we develop the class diagram according to some cases introduces in our use case diagram with some modifications. Also we propose attributes for our classes. The class diagram for our project is as followed:



Another thing that was done for the second iteration was developing CRUD analysis. CRUD means create, read, update and delete. Crud matrix is a table that its rows are our use cases and its columns are the classes of our system. Below the CRUD matrix of our system is presented:

CRUD Matrix (C=Create, R=Read, U=Update, D=Delete)	Manager	Engineer	Operator (Control)	Operator (Maintenance)	Customer	Product	Robot	Conveyor	Feeder
Inspecting quality of final product	R	R	CRU			R			
Checking position of the product in work cell	R	R	CRU			R	R	R	R
Checking function of robot and conveyor remotely (INICO)	R	R	CRU				R		R
Programming the robots and conveyors	R	CRU					R	R	
Deciding what feeders to use	R	R	CRU						R
Monitoring condition of sensors and safety swithches	R	R	CRU						
Maintenance of robots, conveyors and feeders	R	R		RU		R		R	R
Defining specifications (models, color)	R	CRU							
Receives order	CRUD								
Checks the ware house	R								
Orders needed material	CRUD								
Paying for ongoing expenses	CRUD								
Creates schedule for workers	CRUD								
Payment	R				CRU				

And last but not least, for the second iteration, we created our first UI using Node.js and with the help of JavaScript. For this case, we first created an SVG for our work cell and using JavaScript we created a server on our machine in order to be able to visualize our UI. For our UI we created pallets as variables to be able to locate them in different positions on our conveyor. The JavaScript code for this case is presented:

```
var http = require('http');

var stop1 = true;
var stop2 = false;
var stop3 = true;
var stop4 = true;

var server = http.createServer(function (request, response) {
  response.writeHead(200, {"Content-Type": "text/html"});
  response.write("<html>");
     response.write("<header>");
     response.write("</header>");
     response.write("<body>");
     response.write("<h1>First UI</h1>");
     response.write("<h2>FASTory Simulator - WS6</h2>");
     response.write("<h3>Group 9: Seyedsina, Rongwei, Enbo</h3>");
     response.write("</body>");
     response.write("</html>");
  response.write("<svg width=\"10000\" height=\"10000\">");

//pallets//
    if (stop1) {
        response.write("<polygon points=\"430,100 440,80 430,60 410,60 400,80 410,100\" style=\"fill:black;stroke:black;stroke-width:1\" />");
    }
    if (stop2) {
        response.write("<polygon points=\"325,100 335,80 325,60 305,60 295,80 305,100\" style=\"fill:black;stroke:black;stroke-width:1\" />");
    }
    if (stop3) {
        response.write("<polygon points=\"280,100 290,80 280,60 260,60 250,80 260,100\" style=\"fill:black;stroke:black;stroke-width:1\" />");
    }
    if (stop4) {
        response.write("<polygon points=\"240,100 250,80 240,60 220,60 210,80 220,100\" style=\"fill:black;stroke:black;stroke-width:1\" />");
    }
```

And the SVG for our UI is:

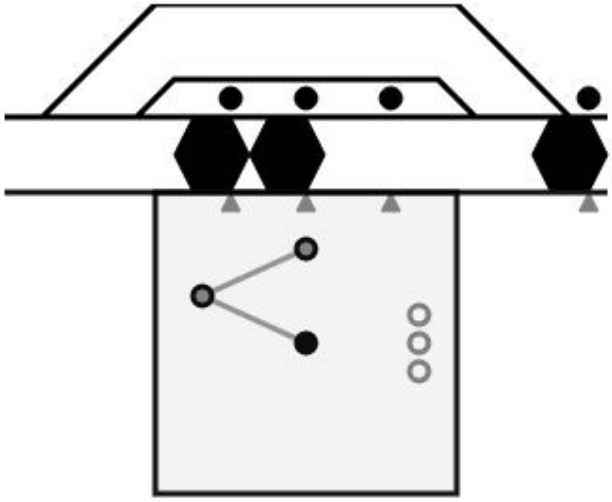
```
  //Sensor//
  response.write("<circle cx=\"430\" cy=\"50\" r=\"6\" stroke=\"black\" stroke-width=\"1\" fill=\"black\" />");
  response.write("<circle cx=\"325\" cy=\"50\" r=\"6\" stroke=\"black\" stroke-width=\"1\" fill=\"black\" />");
  response.write("<circle cx=\"280\" cy=\"50\" r=\"6\" stroke=\"black\" stroke-width=\"1\" fill=\"black\" />");
  response.write("<circle cx=\"240\" cy=\"50\" r=\"6\" stroke=\"black\" stroke-width=\"1\" fill=\"black\" />");

  //Stopper//
  response.write("<polygon points=\"430,100 435,110 425,110\" style=\"fill:grey;stroke:grey;stroke-width:1\" />");
  response.write("<polygon points=\"325,100 320,110 330,110\" style=\"fill:grey;stroke:grey;stroke-width:1\" />");
  response.write("<polygon points=\"280,100 285,110 275,110\" style=\"fill:grey;stroke:grey;stroke-width:1\" />");
  response.write("<polygon points=\"240,100 235,110 245,110\" style=\"fill:grey;stroke:grey;stroke-width:1\" />");

  //Feeder//
  response.write("<circle cx=\"340\" cy=\"165\" r=\"5\" stroke=\"grey\" stroke-width=\"3\" fill=\"white\" />");
  response.write("<circle cx=\"340\" cy=\"180\" r=\"5\" stroke=\"grey\" stroke-width=\"3\" fill=\"white\" />");
  response.write("<circle cx=\"340\" cy=\"195\" r=\"5\" stroke=\"grey\" stroke-width=\"3\" fill=\"white\" />");

  //Robot//
  response.write("<line x1=\"280\" y1=\"180\" x2=\"225\" y2=\"155\" style=\"stroke:rgb(151,151,151);stroke-width:3\" />");
  response.write("<line x1=\"225\" y1=\"155\" x2=\"280\" y2=\"130\" style=\"stroke:rgb(151,151,151);stroke-width:3\" />");
  response.write("<circle cx=\"280\" cy=\"180\" r=\"5\" stroke=\"black\" stroke-width=\"3\" fill=\"black\" />");
  response.write("<circle cx=\"225\" cy=\"155\" r=\"5\" stroke=\"black\" stroke-width=\"3\" fill=\"grey\" />");
  response.write("<circle cx=\"280\" cy=\"130\" r=\"5\" stroke=\"black\" stroke-width=\"3\" fill=\"grey\" />");

  //Convey Line//
  response.write("<polyline points=\"440,60 420,60 360,0 200,0 140,60 190,60 210,40 350,40 370,60 420,60 120,60\" style=\"fill:none;stroke:bla
  response.write("<line x1=\"440\" y1=\"100\" x2=\"120\" y2=\"100\" style=\"stroke:rgb(21,21,21);stroke-width:3\" />");

  //WS 8 Working Station//
  response.write("<rect x=\"200\" y=\"100\" width=\"160\" height=\"160\" style=\"fill:grey;stroke:black;stroke-width:3;fill-opacity:0.1;stroke
  response.end();
});
```

Now we can run the code on our server and check the results there:

First UI

FASTory Simulator - WS6

Group 9: Seyedsina, Rongwei, Enbo



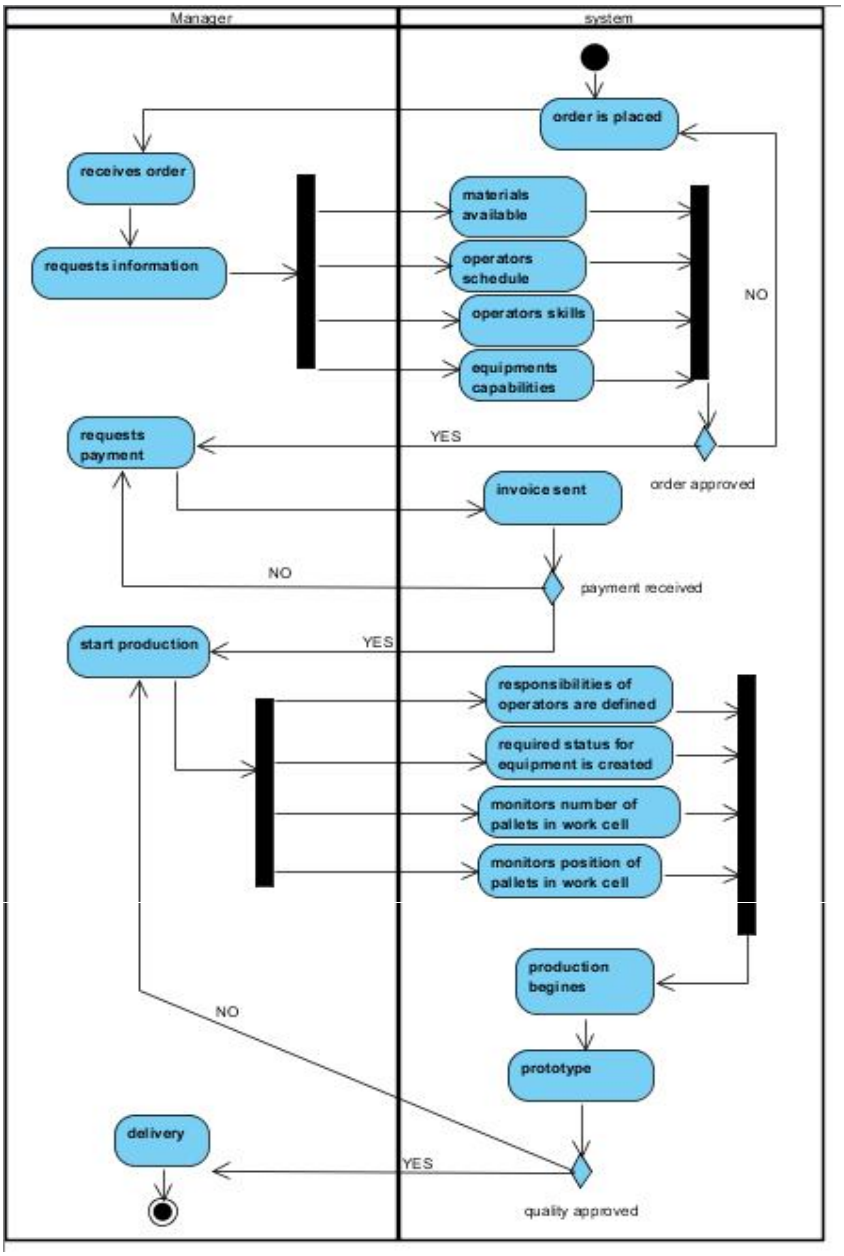
This is how our first UI look like!

Third iteration with modified diagrams

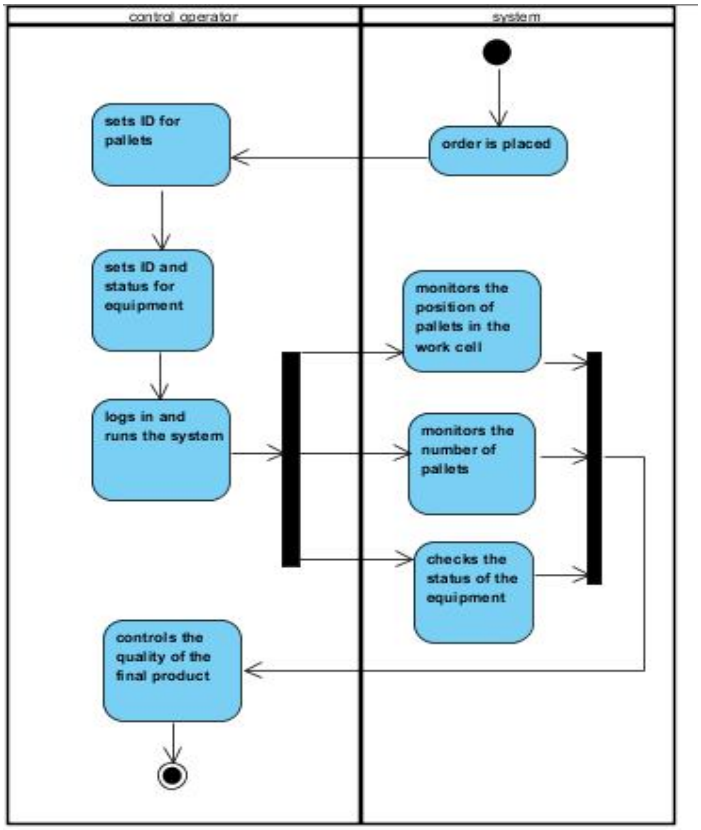
For the third iteration, we created activity diagram, sequence diagram and state chart.

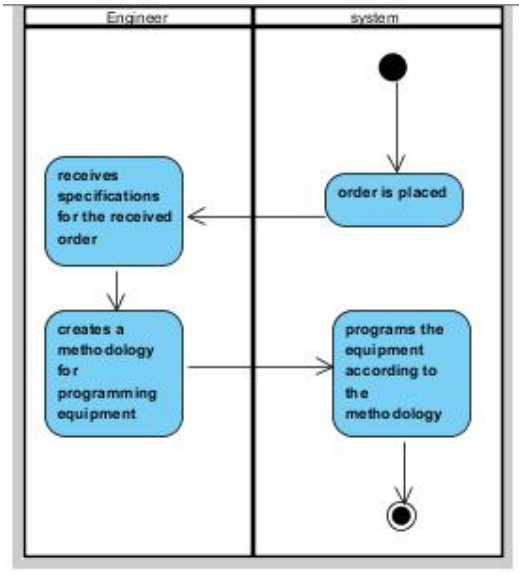
Activity diagram is basically a flow chart to represent the flow form one activity to another activity. The activity can be described as an operation of the system. For our system, we have created activity diagram for correlation between different actors and our system. Our activity diagram is presented below:

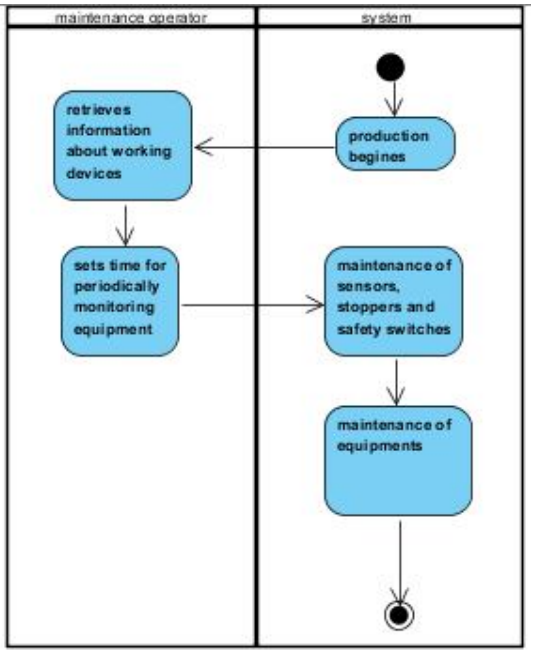
This one is activity diagram between manager and system:



The next ones are activity diagrams for engineer, control operator and maintenance operator with system:

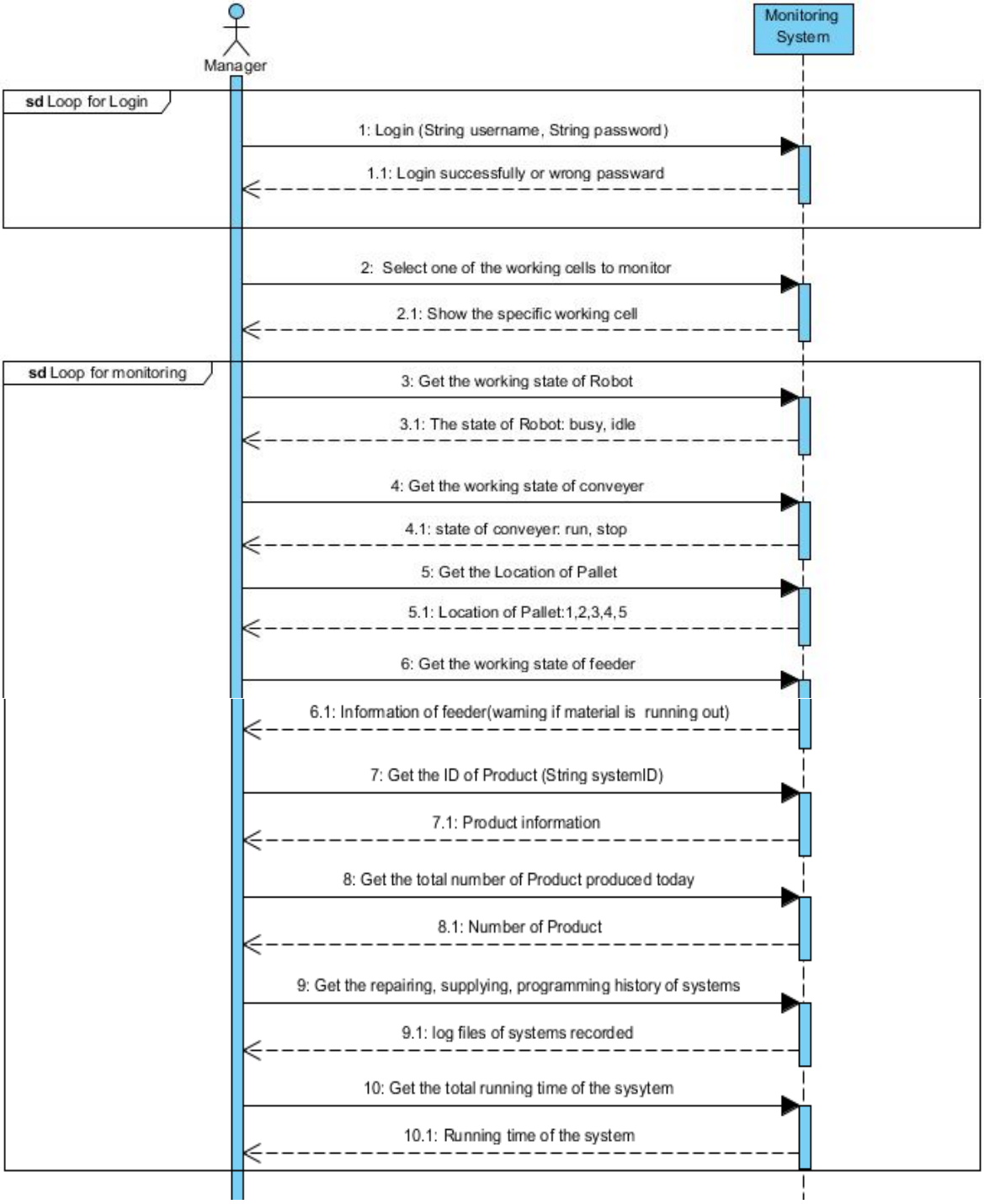


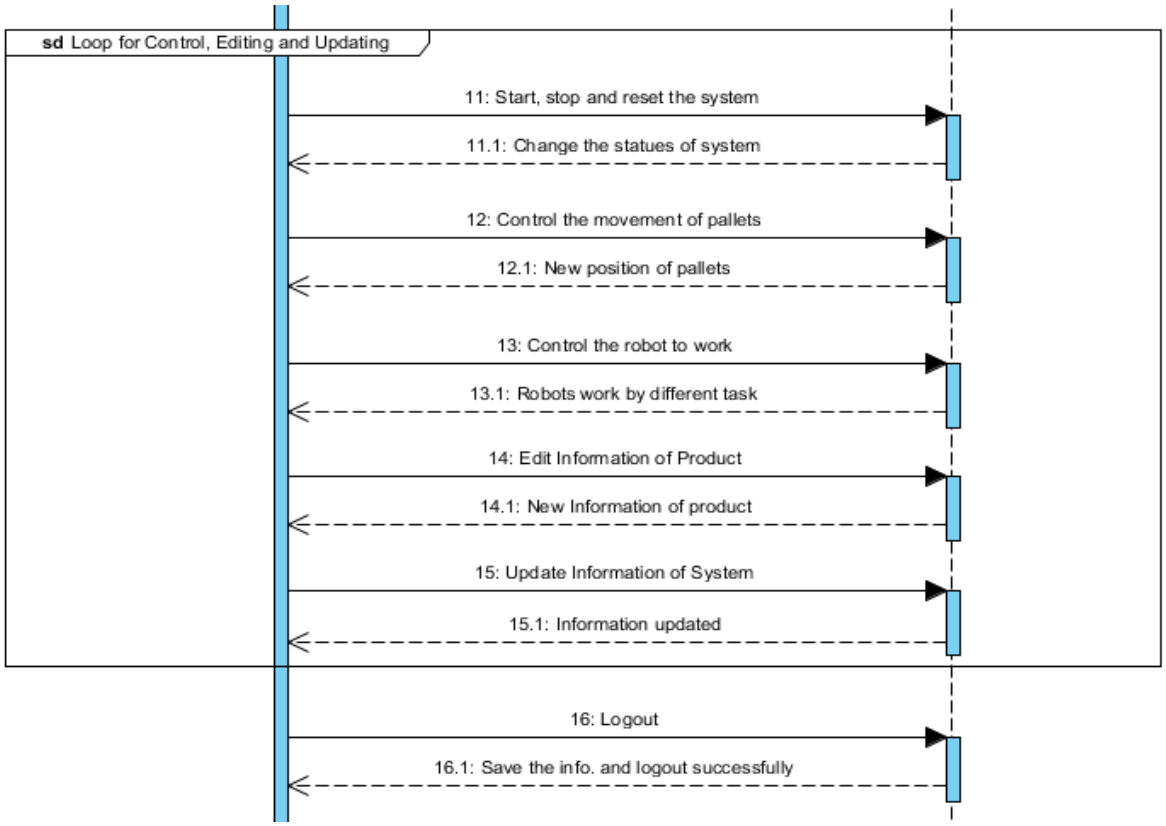




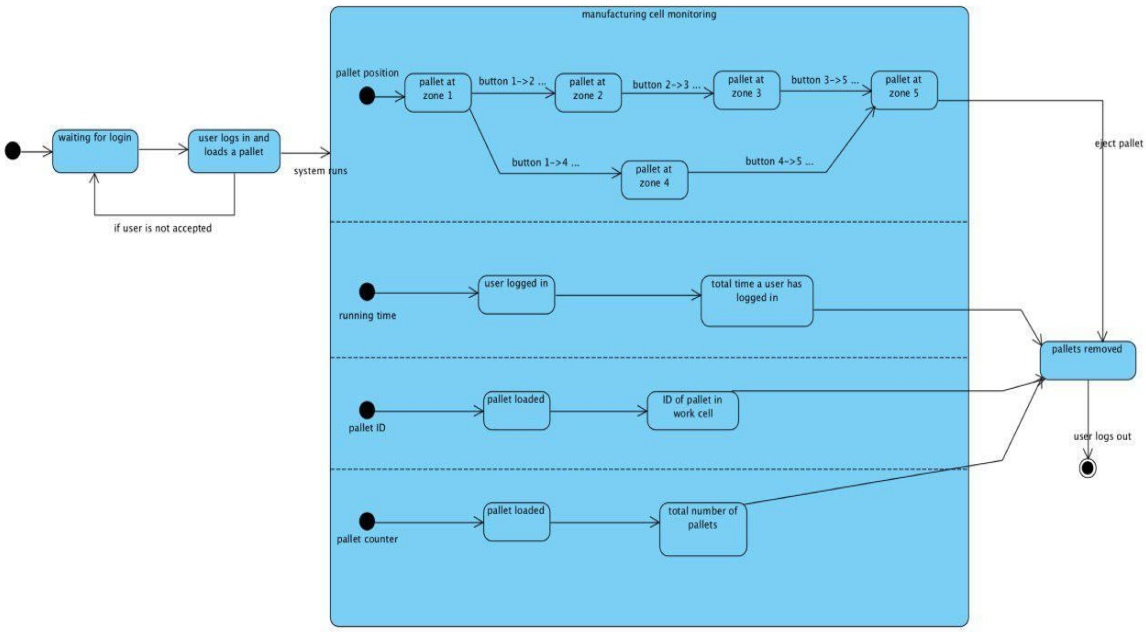
Sequence diagram shows the interactions between our actor (manager) and the monitoring system.

For our case this diagram is as follows:





And the last diagram for this iteration was state chart. State chart describes different states of a component in a system. The states are specific to a component/object of a system. For our system we have proposed the states in our monitoring process and created a state chart based on that. Our state chart is as follows:



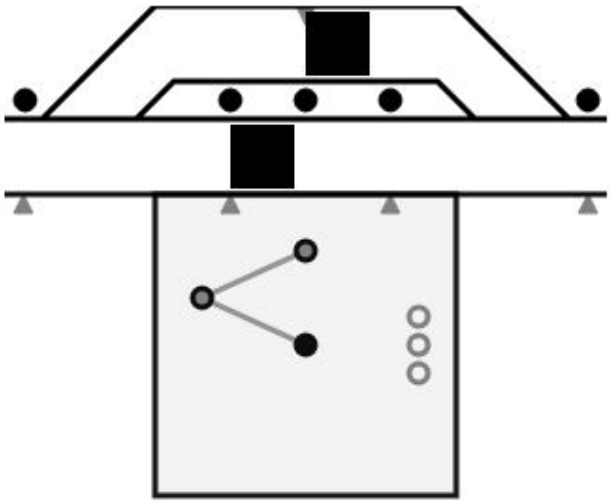
And finally for the third iteration we developed our first web server implementation and created animation for our previously made SVG. In order to create our web server implementation, in a server.js note pad, we created program for running a server and installed needed modules such as express and node-rest client. Then we placed our SVG in an HTML code notepad and using the server.js file, we created buttons for transitions in our animation on our web server.

A screenshot of our web server is as follows:

First Animation

FASTory Simulator - WS6

Group 9: Seyedsina, Rongwei, Enbo



Insert Pallet
Move 1->2
Move 2->3
Move 3->5
Move 1->4
Move 4->5
Eject Pallet
Clear system

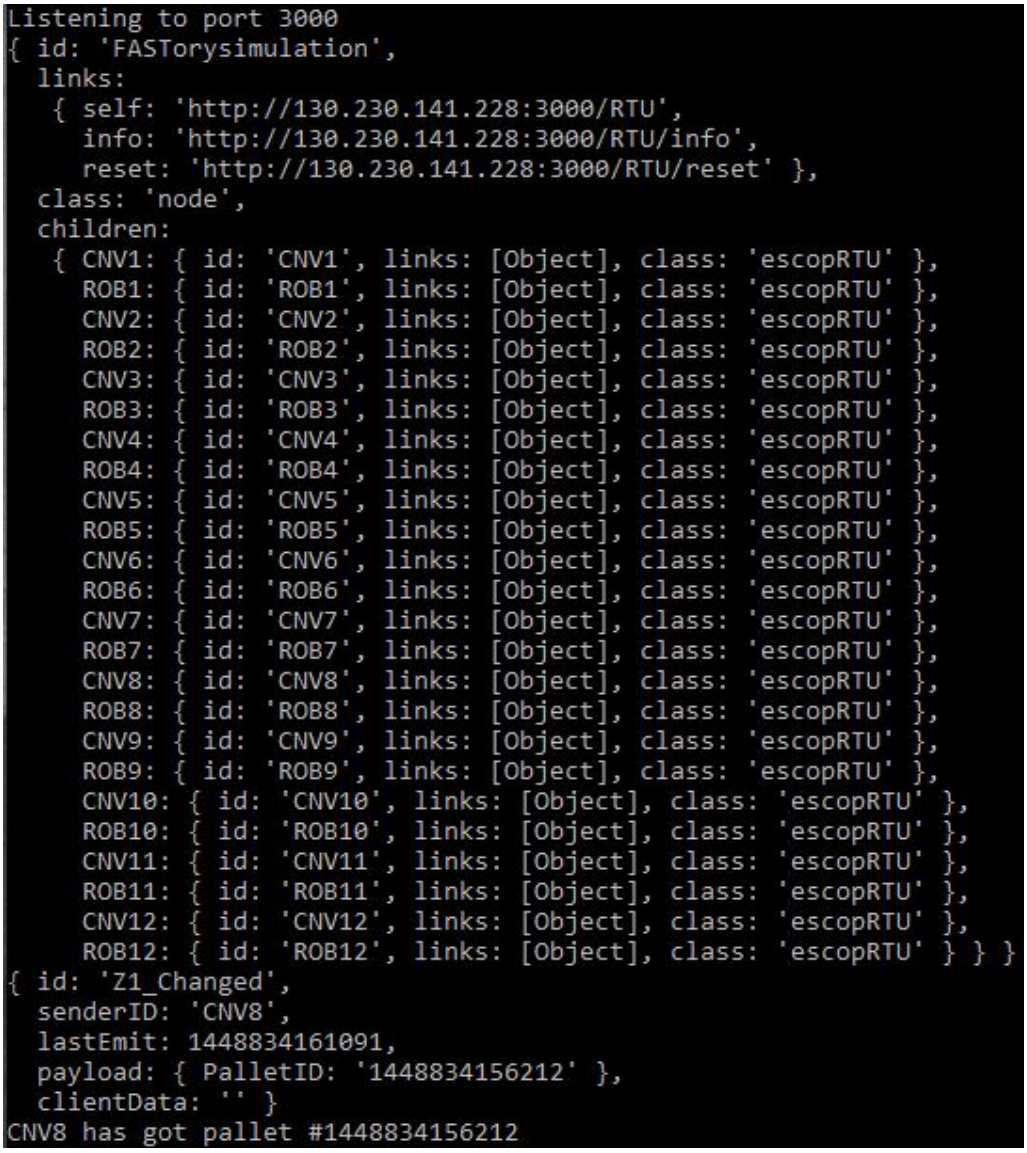
Fourth iteration

For our fourth iteration, we first refined all the previously made diagrams which are presented in this report.

The next task was to connect the web server to the FASTory simulator. For this case we first added Parser module to the previously installed module. After that we created a server and connected it to the FASTory simulator. Then, we modified the server so that be able to respond when the simulator is invoked. Below there is a code for our server implementation:

```
var express = require('express');
    app = express();

    var bodyParser = require('body-parser') //analyse the responses
    app.use(bodyParser.json());//use json format

    var Client = require('node-rest-client').Client; //creat client
client = new Client();

var args = {
    data: {"destUrl": "http://130.230.158.176:3000/"},  //Here is my own IP address
    headers:{"Content-Type": "application/json"}
};

/*Subscribe the notification of zone 1~5 */

client.get("http://escop.rd.tut.fi:3000/RTU", function(data, response){ //check system is online
    console.log(JSON.parse(data));
    client.post("http://escop.rd.tut.fi:3000/RTU/CNV8/events/Z1_Changed/notifs", args, function(data,response) {
    });
    client.post("http://escop.rd.tut.fi:3000/RTU/CNV8/events/Z2_Changed/notifs", args, function(data,response) {
    });
    client.post("http://escop.rd.tut.fi:3000/RTU/CNV8/events/Z3_Changed/notifs", args, function(data,response) {
    });
    client.post("http://escop.rd.tut.fi:3000/RTU/CNV8/events/Z4_Changed/notifs", args, function(data,response) {
    });
    client.post("http://escop.rd.tut.fi:3000/RTU/CNV8/events/Z5_Changed/notifs", args, function(data,response) {
    });
});

app.get('/', function(req,res){ //routes http get requests
    res.send('hello world');
});

app.post('/', function(req,res){ //routes http post requests
    console.log(req.body);
    console.log(req.body.senderID + ' has got pallet #' + req.body.payload.PalletID);//get notification once the pallet Id in zone1~5
    res.send('{}');
});

app.listen(3000, function(){
    console.log('Listening to port 3000');
});
```

The image below represents the respond we get on our terminal after a transition in simulator is invoked:



Next, we created objects according to the classes we had. The objects that we have considered are: robot, pallet and conveyor. These objects are created according to our classes in the class diagram.

The screenshots below represent the codes for our objects and contain additional comments for further clarification:

The first object is our pallet:

```
/* Make pallet as an object*/
var Pallet = function (palletId, productType, palletLocation) {
    this._palletId = palletId;
    this._productType = productType;
    this._palletLocation = palletLocation;
}
/* Define the prototype of pallet*/
Pallet.prototype = {

/*get Id of pallet*/
    get_palletId : function(){
        return this._palletId;
    },
    set_palletId : function(palletId){
        this._palletId = palletId;
    },

/*get and set type of product*/
    get_productType : function(){
        return this._productType;
    },
    set_productType : function(productType){
        this._productType = productType;
    },

/*get and set type of palletlocation*/
    get_palletLocation : function(){
        return this._palletLocation;
    },
    set_palletLocation : function(palletLocation){
        this._palletLocation = palletLocation;
    },

    to_str: function () {
        return JSON.stringify(this);
    }
}
module.exports = Pallet;
```

Next object is our robot:

```
/* Make robot as an object*/
var Robot = function (robotId, robotStatus) {
    this._robotId = robotId;
    this._robotStatus = robotStatus;
}

/* Define the prototype of robot*/
Robot.prototype = {

/*get and set Id of robot*/
    get_RobotId : function(){
        return this._robotId;
    },
    set_RobotId : function(RobotId){
        this._robotId = robotId;
    },

/*get and set status of robot*/
    get_RobotStatus : function(){
        return this._robotStatus;
    },

    set_RobotStatus : function(RobotId){
        this._robotStatus = robotStatus;
    },

    to_str: function () {
        return JSON.stringify(this);
    }

}

module.exports = Robot;
```

And the last object we created is conveyor:

```
/*import pallet module*/
var Pallet = require('./pallet.js');
/*make conveyor as an object*/
var Conveyor = function(){
    this._pallets = [];
    this._zones = {
        "z1" : {},
        "z2" : {},
        "z3" : {},
        "z4" : {},
        "z5" : {}
    };
}
/* Define the prototype of conveyor*/
Conveyor.prototype = {

/* insert a new pallet in zone1 */
    insert_pallet: function(zone, PalletId){
        if(zone in this._zones){
            var p = new Pallet("defaultID", PalletId);
            this._zones[zone] = x;
            this._pallets.push(x);
        }else{
            console.log('No such zone');
        }
    },

/* move the pallet from zone1 to zone2 if there is a pallet in zone1 */
    TransZone12: function(){
        if(this._zones in zone1){
            this._PalletId=PalletId;
            this._zones[zone] = zones[x+1];
            this._pallets.push(x+1);
        }else{
            console.log('No pallet in the zone 1');
        }
    },

/* move the pallet from zone2 to zone3 if there is a pallet in zone2 */
    TransZone23: function(){
        if(this._zones in zone2){
            this._PalletId=PalletId;
            this._zones[zone] = x+1;
            this._pallets.push(x+1);
        }else{
            console.log('No pallet in the zone 2');
        }
    },
/* move the pallet from zone3 to zone5 if there is a pallet in zone3 */
    TransZone35: function(){
        if(this._zones in zone3){
            this._PalletId=PalletId;
            this._zones[zone] = x+2;
            this._pallets.push(x+2);
        }else{
            console.log('No pallet in the zone 3');
        }
    },

```

```
/* move the pallet from zone1 to zone4 if there is a pallet in zone1 */
    TransZone14: function(){
        if(this._zones in zone1){
            this._PalletId=PalletId;
            this._zones[zone] = x+3;
            this._pallets.push(x+3);
        }else{
            console.log('No pallet in the zone 1');
        }
    },

/* move the pallet from zone4 to zone5 if there is a pallet in zone4 */
    TransZone45: function(){
        if(this._zones in zone4){
            this._PalletId=PalletId;
            this._zones[zone] = x+1;
            this._pallets.push(x+1);
        }else{
            console.log('No pallet in the zone 4');
        }
    },

/* get type of product in the conveyor */
    get_product_type_in_conveyor: function () {

        var ids = [];

        var i;
        for (i = 0; i< this._pallets.length; i++ ){

            var newId = this._pallets[i].get_productType();
            if(newId !== -1){
                ids.push(newId);
            }
        }
        return ids;
    }
}
module.exports = Conveyor;
```

Final iteration

Programming codes

Server code

Firstly, we defined the parameters. Here we use 'express' frame and the language between front end and back end are same. Then we create the local server and insert socket package. After those steps, we use 'get' method to catch the variety from the animation website and we send the '/index.html' categories to it.

```
19  var express = require('express');
20  var app = express();
21  var server = require('http').createServer(app);
22  var io = require('socket.io')(server);
23
24  //Defining a route from localhost:3000/ to ./client/index.html
25  app.get('/', function(req, res){
26      res.sendFile(__dirname + '/index.html');
27  });
28
29  //Defining resource filter from /js/** to ./client/js/**
30  app.use('/js', express.static(__dirname +'/js'));
31
```

From line 33 we start to monitor the website. The server port is 3000. We analyse the responses by using 'json' format. Then we create new client and get the message and storage in my own IP address. In our case "http://130.230.159.106:3000/" has been used as our IP address for as our desUrl.

```
33  //Running server on port 3000. On started executin logging function.
34  //A typical example of async function call.
35  server.listen(3000, function(){
36      console.log('Started on port 3000...');
37  });
38
39
40  var bodyParser = require('body-parser') //analyse the responses
41  app.use(bodyParser.json());//use json format
42
43  var Client = require('node-rest-client').Client; //creat client
44  client = new Client();
45
46  var args = {
47      //data: {"destUrl": "http://localhost:3000/"},  //Here is my own IP addre
48
49      data: {"destUrl": "http://130.230.159.106:3000/"},  //Here is my own IP a
50      headers:{"Content-Type": "application/json"}
51  };
52
```

Then the client check the system online, if there is data changing then the client will collect it and transfer to the JSON format, which is easier understanding. Then we create five different zones for five different point in the factory line. Then it can be transferred to the server and displayed.

```
/*Subscribe the notification of zone 1~5 */

client.get("http://escop.rd.tut.fi:3000/RTU", function(data, response){ //check system is online
    //console.log(JSON.parse(data));
    client.post("http://escop.rd.tut.fi:3000/RTU/CNV8/events/Z1_Changed/notifs", args, function(data,response) {
    });
    client.post("http://escop.rd.tut.fi:3000/RTU/CNV8/events/Z2_Changed/notifs", args, function(data,response) {
    });
    client.post("http://escop.rd.tut.fi:3000/RTU/CNV8/events/Z3_Changed/notifs", args, function(data,response) {
    });
    client.post("http://escop.rd.tut.fi:3000/RTU/CNV8/events/Z4_Changed/notifs", args, function(data,response) {
    });
    client.post("http://escop.rd.tut.fi:3000/RTU/CNV8/events/Z5_Changed/notifs", args, function(data,response) {
    });
});
```

Here we continue defining the variable and inserting socket function.

```
//var palletNumber = 0;
var data;
var time=0;
var locationID = '';
var mySocket;
var palletIDList = [];
io.on('connection', function(socket){
    mySocket = socket;
});
```

Then we get the notification if the pallet ID changes in zone 1 to 5. We create a new route for the ’/’ path. The ‘data’, which is the pallet ID number is passed through the ‘body’ property in the ‘req ‘object.

```
app.post('/', function(req,res){ //routes http post requests
    console.log(req.body);
    data = req.body;
    console.log(req.body.senderID + ' has got pallet #' + req.body.payload.PalletID);//get


    //if(time % 500 == 0) {
    //
    //  console.log('%d-- locationID= %s, bodyID = %s', time/500, locationID, data.id);
    //}
    // if the pallet location changed, then send data to client
```

If the location ID is not same with Pallet ID, which is also not equal to -1, the socket emit the Pallet ID and location to the client. The location ID cannot be -1 since there is no pallet the location ID is -1 and the socket will emit the data, which we are not expecting.

```
    if (locationID != data.id && req.body.payload.PalletID != -1 ) {

        //palletIDList.foreach(function(element){
        //  if (element != req.body.payload.PalletID) {
        //      palletIDList.push(req.body.payload.PalletID);
        //  }
        //});

        //palletNumber++;
        mySocket.emit('palletData', {
            senderID: data.senderID,
            location: data.id,
            palletID: data.payload.PalletID
            //palletNumber: palletNumber
        });
        locationID = data.id;

    }
```

Here we define the simulator address, which is "http://escop.rd.tut.fi:3000/".

```
var service_body_args = {
    //data: {"destUrl": server_IP},  //Here is server IP address

    data: {"destUrl": "http://escop.rd.tut.fi:3000/"},  //Here is server IP address
    headers:{"Content-Type": "application/json"}
};
```

Then the client sends the data to server to make the corresponding changes. Here we use socket to receive data from client. There is a setTimeout function for waiting the "app" document send emit first. Then client sends data to the server. If the request is transfer zone 12 then the pallet moves from position 1 to 2. When it is transfer zone 23 the pallet moves from position 2 to 3. And the rest as follow.

```
// send data to server
client.post('http://escop.rd.tut.fi:3000/fmw', function(req, res){

    io.on('connection', function(socket){

        socket.on('LoadPaper', function(){
            client.post('http://130.230.141.228:3000/RTU/ROB7/services/LoadPallet', service_body_args, function(data, response){});
            setTimeout(function(){
            client.post('http://130.230.141.228:3000/RTU/CNV7/services/TransZone35', service_body_args, function(data, response){});
            },10);
        });

        socket.on('TransZone12', function(){
            client.post('http://130.230.141.228:3000/RTU/CNV8/services/TransZone12', service_body_args, function(data, response){});
        });

        socket.on('TransZone23', function(){
            client.post('http://130.230.141.228:3000/RTU/CNV8/services/TransZone23', service_body_args, function(data, response){});
        });

        socket.on('TransZone35', function(){
            client.post('http://130.230.141.228:3000/RTU/CNV8/services/TransZone35', service_body_args, function(data, response){});
        });

        socket.on('TransZone14', function(){
            client.post('http://130.230.141.228:3000/RTU/CNV8/services/TransZone14', service_body_args, function(data, response){});
        });

        socket.on('TransZone45', function(){
            client.post('http://130.230.141.228:3000/RTU/CNV8/services/TransZone45', service_body_args, function(data, response){});
        });
    });
});
```

App code

Here we define the socket, pallet, Data, start time, change date and change time. We also set a palletIDlist to store all the pallet ID, which is array type.

```
var socket = io();
var pallet_id = '';

var startData = new Date();
var startTime = startData.getTime(startData);
var changedDate;
var changedTime;

var palletIDlist = [];
//var palletlist = [];
//var counter = 0;
//var toPosition = {};
//var fromPosition = {};
var addedpallet = false;
var hasExist;
var i;
```

Here we get the data from server and we need to analysis and give the corresponding response. First, we judge if the pallet already exists or not. If the pallet does not exist, then we will execute next if function. We compare the pallet number in the palletIDlisht. When the palletIDlist is equal to the palletID we receive from server (not only the number but also the format), the hasExist is true, which means no need to add more pallet.

```
socket.on('palletData', function (msg) {


    //console.log('palletIDlist: %s', palletIDlist);
    //console.log('palletIDlist.length: %s',palletIDlist.length);
     hasExist = false;
    // to decide whether the pallet has been added or not
    for ( i=0; i < palletIDlist.length; i++){
       if (palletIDlist[i] == msg.palletID){
           // if the new pallet exists, then set hasExist True
           hasExist = true;
       }
    }
```

However, when the hasExist is not ture, it will add new pallet and display the pallet ID on page..

```
        // if the new pallet is not added, then add a new pallet
        if (!hasExist) {                    //不一样就要添加
            pallet_id = msg.palletID;

            // add a new palllet
            add_pallet1();

            console.log('add pallet');
            addedpallet = true;

            palletIDlist.push(msg.palletID);

            // display the palletID on page
            $('#palletID').append($('<li>').text(pallet_id));
```

Besides, it also shows the total number of added pallets, which is the palletIDlist length. Then update the running time after every second. In the end, we build interval to change date and time. In line 64 is setting the hours time, line 67 builds the minutes value and line 70 for seconds. As we want to know the time since start implementing system, we use change time minus start time. Then they are displayed in our UI.

```
51              // display the total number of added pallets
52              document.getElementById("palletNumber").innerHTML = palletIDlist.length;
53              // update the running time after every second
54              setInterval(function (){
55                  changedDate = new Date();
56                  changedTime = changedDate.getTime(changedDate);
57
58                  var difference = changedTime - startTime;
59
60
61                  //var daysDifference = Math.floor(difference/1000/60/60/24);
62                  //difference -= daysDifference*1000*60*60*24;
63
64                  var hoursDifference = Math.floor(difference/1000/60/60);
65                  difference -= hoursDifference*1000*60*60;
66
67                  var minutesDifference = Math.floor(difference/1000/60);
68                  difference -= minutesDifference*1000*60;
69
70                  var secondsDifference = Math.floor(difference/1000);
71
72                  //time = new Date(changedTime - startTime);
73
74                  display = hoursDifference+':'+minutesDifference+':'+secondsDifference;
75                  // calculate the running time
76                  //display = (time.getHours()-2)+':'+time.getMinutes()+':'+time.getSeconds();
77                  document.getElementById("time").innerHTML = display;
78              }, 1000);
79
80          }
81      //}
```

If the factory simulation page changes, the UI should have relevant movement. For example, the pallet move from location 1 to 2, here will implement pallet move from 1 to 2 in the UI and so on. the rest from pallet 2 to 3, 1 to 4, 3 to 5 and 4 to 5 are the same.

```
75          if (msg.location == 'Z2_Changed') {
76              move_pallet12();
77          } else if (msg.location == 'Z3_Changed') {
78              move_pallet23();
79          } else if (msg.location == 'Z4_Changed') {
80              move_pallet14();
81          } else if (msg.location == 'Z5_Changed') {
82              move_pallet35();
83              move_pallet45();
84          }
```

When we detect the changing on UI page, we emit the operation to server on simulation. The task include insert Pallet, move pallet from 1 to 2, 2 to 3, 1 to 4, 3 to 5 and 4 to 5

```
94      function insertPallet() {
95
96          socket.emit('LoadPaper', 'LoadPaper');
97
98
99          //
100         //add_pallet1();
101         //addedpallet = true;
102         //counter++;
103         //socket.emit('LoadPaper', 'LoadPaper');
104         //palletlist.push(document.getElementById('pallet' + counter));
105         //
106         //toPosition.x = config.positions[0].x;
107         //toPosition.y = config.positions[0].x;
108         ////console.log(" palletX: %s; palletY :%s", palletlist[counter - 1].getAttribute('x'), palletlist[counter - 1].getAttribute('y'));
109         //
110
111     }
```

The green codes are the previous codes we created, we want to transfer the pallet base on the coordinates. Since the upper and left has the minimum number of x and y.

```
113     function move12() {
114         socket.emit('TransZone12', 'TransZone12');
115
116         //if (addedpallet) {
117         //    toPosition.x = config.positions[1].x;
118         //    toPosition.y = config.positions[1].y;
119         //    fromPosition.x = palletlist[counter - 1].getAttribute('x');
120         //    fromPosition.y = palletlist[counter - 1].getAttribute('y');
121         //
122         //    //console.log(" fromPositionX: %s; fromPositionY :%s", fromPosition.x, fromPosition.y);
123         //    //console.log(" toPositionX: %s; toPositionY :%s", toPosition.x, toPosition.y);
124         //
125         //    if (toPosition.x < fromPosition.x) {
126         //        move_pallet12();
127         //        socket.emit('TransZone12', 'TransZone12');
128         //    }
129         //}
130
131     }
```

```
133     function move23() {
134
135         socket.emit('TransZone23', 'TransZone23');
```

```
154     function move35() {
155         socket.emit('TransZone35', 'TransZone35');
```

```
174     function move14() {
175         socket.emit('TransZone14', 'TransZone14');
```

```
195     function move45() {
196         socket.emit('TransZone45','TransZone45');
```

Animation

Below are the code how we defined the position according to the screen. The pallet size is 34 in this project. The position is based on the principle that left and up have the smallest value.

```
var config = {
    pallet : {
        size : 34
    },
    positions : [
        {
            x:430,
            y:63
        },
        {
            x:325,
            y:63
        },
        {
            x:240,
            y:63
        },
        {
            x:280,
            y:3
        },
        {
            x:130,
            y:63
        }
    ]
}
```

```
var pallets = [1,1,1,1,1];
var pallet;
var n = 0;
function add_pallet1(){
    n++;
    var svg = d3.select('svg');
    pallet = svg.append('rect')
        .attr("x" , config.positions[0].x)
        .attr("y" , config.positions[0].y)
        .attr("width" , config.pallet.size)
        .attr("height" , config.pallet.size);
};
```

Index.html

The index.html file is using for creating the html of our UI. Here we use socket, jquery and also import d3,animation.js and app.js:

```
<!DOCTYPE html>
<html>
<head>
    <script src="https://cdn.socket.io/socket.io-1.2.0.js"></script>
    <script src="http://code.jquery.com/jquery-1.11.1.js"></script>
    <script src="js/d3.js"></script>
    <script src="js/animation.js"></script>
    <script src="js/app.js"></script>
```

Here we refine the buttons UI by CSS Button Generator(http://css3buttongenerator.com/)

```
    <style>


        #buttonsDiv .btn {

            margin: 3px;
            background: #dbddde;
            background-image: -webkit-linear-gradient(top, #dbddde, #b3b7ba);
            background-image: -moz-linear-gradient(top, #dbddde, #b3b7ba);
            background-image: -ms-linear-gradient(top, #dbddde, #b3b7ba);
            background-image: -o-linear-gradient(top, #dbddde, #b3b7ba);
            background-image: linear-gradient(to bottom, #dbddde, #b3b7ba);
            -webkit-border-radius: 10;
            -moz-border-radius: 10;
            /*border-radius: 10px;*/
            font-family: Georgia;
            color: #000000;
            font-size: 16px;
            padding: 5px 10px 5px 10px;
            text-decoration: none;

            width: 130px;
            height: 40px;;
        }
```

```
        #buttonsDiv .btn:focus, .btn:active, .btn.active, .btn:focus:active {
            background-image: none;
            outline: none;
            -webkit-box-shadow: none;
            box-shadow: none;
        }

        #buttonsDiv .btn:hover {
            background: #c2bac2;
            background-image: -webkit-linear-gradient(top, #c2bac2, #b1b4b5);
            background-image: -moz-linear-gradient(top, #c2bac2, #b1b4b5);
            background-image: -ms-linear-gradient(top, #c2bac2, #b1b4b5);
            background-image: -o-linear-gradient(top, #c2bac2, #b1b4b5);
            background-image: linear-gradient(to bottom, #c2bac2, #b1b4b5);
            text-decoration: none;
        }

        #buttonsDiv {
            width: 40%;
            float: left;
            background-color: ;
```

```
        }
        #info {
            width: 50%;
            float: left;
        }

        #info .margin {
            margin: 8px;;
        }

    </style>

</head>
```

Here illustrates what is the title for the UI page.

```
<body>

<h1>First Animation</h1>
<h2>FASTory Simulator - WS6</h2>
<h3>Group 9: Seyedsina, Rongwei, Enbo</h3>

<div id="cell">
<svg width="500" height="300">
```

Then we creat the sensor which are the same size with x,y position and radius, balck color, stroke width is equal to 1 and fill in with black color. Then we creat the stopper opposite the sensor by

polygon. Feeder and robot are also created by the samilar way with sensor.

```
<!--sensor-->
    <circle cx="430" cy="50" r="6" stroke="black" stroke-width="1" fill="black" />
    <circle cx="325" cy="50" r="6" stroke="black" stroke-width="1" fill="black" />
    <circle cx="280" cy="50" r="6" stroke="black" stroke-width="1" fill="black" />
    <circle cx="240" cy="50" r="6" stroke="black" stroke-width="1" fill="black" />
    <circle cx="131" cy="50" r="6" stroke="black" stroke-width="1" fill="black" />

<!--stopper-->
    <polygon points="430,100 435,110 425,110" style="fill:grey;stroke:grey;stroke-width:1" />
    <polygon points="325,100 320,110 330,110" style="fill:grey;stroke:grey;stroke-width:1" />
    <polygon points="240,100 235,110 245,110" style="fill:grey;stroke:grey;stroke-width:1" />
    <polygon points="280,10 285,0 275,0" style="fill:grey;stroke:grey;stroke-width:1" />
    <polygon points="130,100 125,110 135,110" style="fill:grey;stroke:grey;stroke-width:1" />
<!--Feeder-->
    <circle cx="340" cy="165" r="5" stroke="grey" stroke-width="3" fill="white" />
    <circle cx="340" cy="180" r="5" stroke="grey" stroke-width="3" fill="white" />
    <circle cx="340" cy="195" r="5" stroke="grey" stroke-width="3" fill="white" />

<!--Robot-->
    <line x1="280" y1="180" x2="225" y2="155" style="stroke:rgb(151,151,151);stroke-width:3" />
    <line x1="225" y1="155" x2="280" y2="130" style="stroke:rgb(151,151,151);stroke-width:3" />
    <circle cx="280" cy="180" r="5" stroke="black" stroke-width="3" fill="black" />
    <circle cx="225" cy="155" r="5" stroke="black" stroke-width="3" fill="grey" />
    <circle cx="280" cy="130" r="5" stroke="black" stroke-width="3" fill="grey" />
```

Then we created one convery line

```
<!--Convey Line-->
    <polyline points="440,60 420,60 360,0 200,0 140,60 190,60 210,40 350,40 370,60 420,60 120,60" style="fill:none;stro
    <line x1="440" y1="100" x2="120" y2="100" style="stroke:rgb(21,21,21);stroke-width:3" />

<!--WS 8 Working Station-->
    <rect  x="200" y="100" width="160" height="160" style="fill:grey;stroke:black;stroke-width:3;fill-opacity:0.1;strok
```

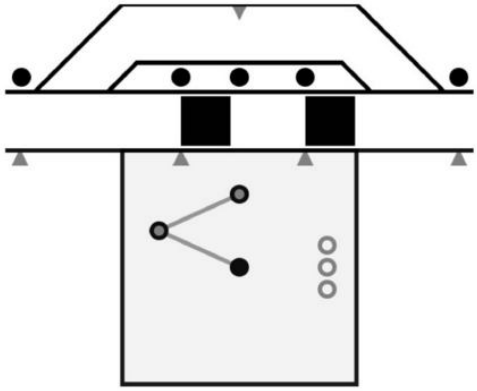
The button 1 is for inserting pallet, button 2 for moving pallet from 1 to 2, button 3 for moving pallet from 2 to 3, button 4 for moving pallet from 3 to 5, button 5 for moving pallet from 1 to 4 and button 6 for moving pallet from 4 to 5.

And we also show our pallet ID , number and running time here:

```
</div>
<div id="buttonsDiv">
    <button class="btn" onclick="insertPallet()"> Insert Pallet </button><br>
    <button class="btn" onclick="move12()"> Move 1->2 </button><br>
    <button class="btn" onclick="move23()"> Move 2->3 </button><br>
    <button class="btn" onclick="move35()"> Move 3->5 </button><br>
    <button class="btn" onclick="move14()"> Move 1->4 </button><br>
    <button class="btn" onclick="move45()"> Move 4->5 </button><br>
    <!--<button class="btn" onclick="ejectPallet()"> Eject Pallet </button><br>-->
    <!--<button class="btn" onclick="clearSystem()"> Clear system </button><br>-->

</div>

<div id="info">

    <div id="pallet_number" class="margin">
        PalletNumber: <span id="palletNumber"> </span>
    </div>

    <div id="running_time" class="margin">
        Running Time: <span id="time"></span>
    </div>
    <div id="pallet_id" class="margin">PalletsID: <span id="palletID"></span></div>

</div>

</body>
</html>
```

Here is the UI page we created. pallets move from right to left and there are totally 5 points and two pathes. The fist path that pallets move from 1 to 2, 2 to 3, 3 to 5. The other path is that pallets move from 1 to 4 and 4 to 5 position. It shows the pallet number, running time and palletsID in ths UI page.

Group 9: Seyedsina, Rongwei, Enbo



Insert Pallet

Move 1->2

Move 2->3

Move 3->5

Move 1->4

Move 4->5

PalletNumber: 2

Running Time: 0:0:21

PalletsID:

- 1449579199014
- 1449585294776

Here is the simulator of FASTory system.

Modification and improving

Previous version

At the beginning, we program the code after giving command, the client will give the information to implement the movement directly without informing the server. After the information arrives to operation, it will give command to server. However, in this case the server will give the feedback again to order the new movement. Then we will have two IDs, which is not what we are expecting.

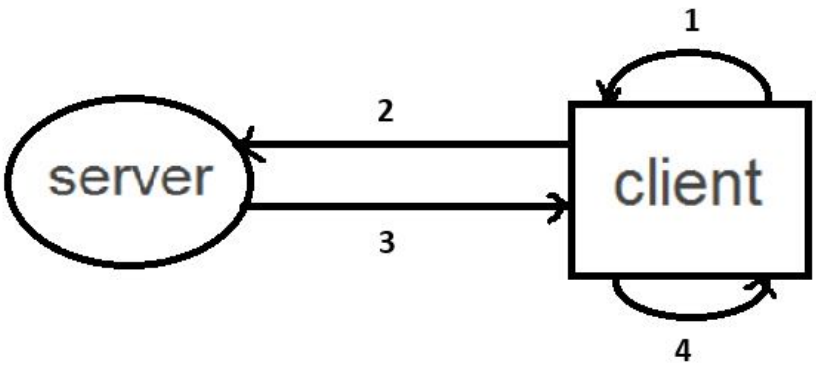


Figure 1 Pervious operation process

Newest version

After improving it, when user operates server, the server sends the command to client directly to active the operation. The figure below illustrates the principle.

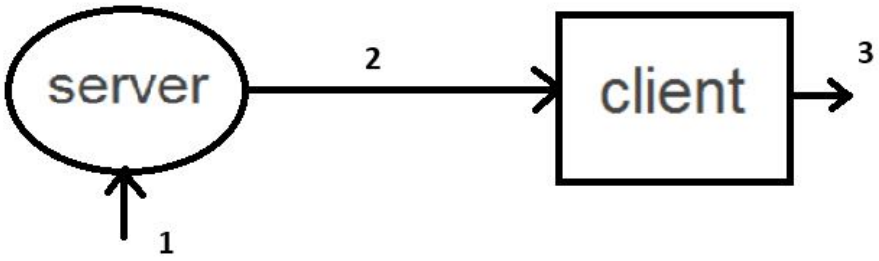


Figure 2 Give command from server

In case client received command, it sends the operation request to server first. Then the server do the corresponding operation according to the client request. Then the client implement operation according to the data given by server.

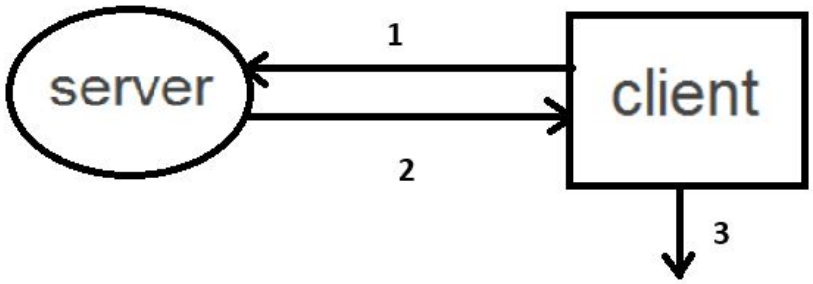


Figure 3 Give command from client

In conclusion, we realize that the system synchronization in this project. The operation we have done in Fastory simulator system could first be detected by server then emit to client so that the UI has the corresponding movement. On the other side, when users implement tasks on UI it could also go through the app file then transfer to server to do the same task in simulator.

This project was a big challenge for us since we had not have background in JavaScript or Node.js. However, we really appreciate that know we understand more how to use JavaScript and node.js for running server and doing web page synchronization.

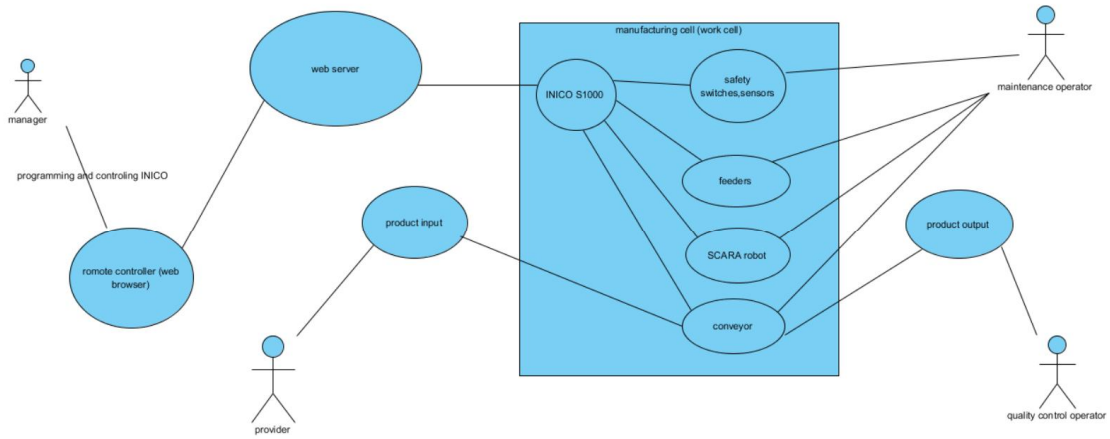
CONCULSIONS

In this section we have represented our progress in diagrams by showing them from first through last iteration. During the iterations our understanding of the whole system has become clearer and this is evident in our diagrams.

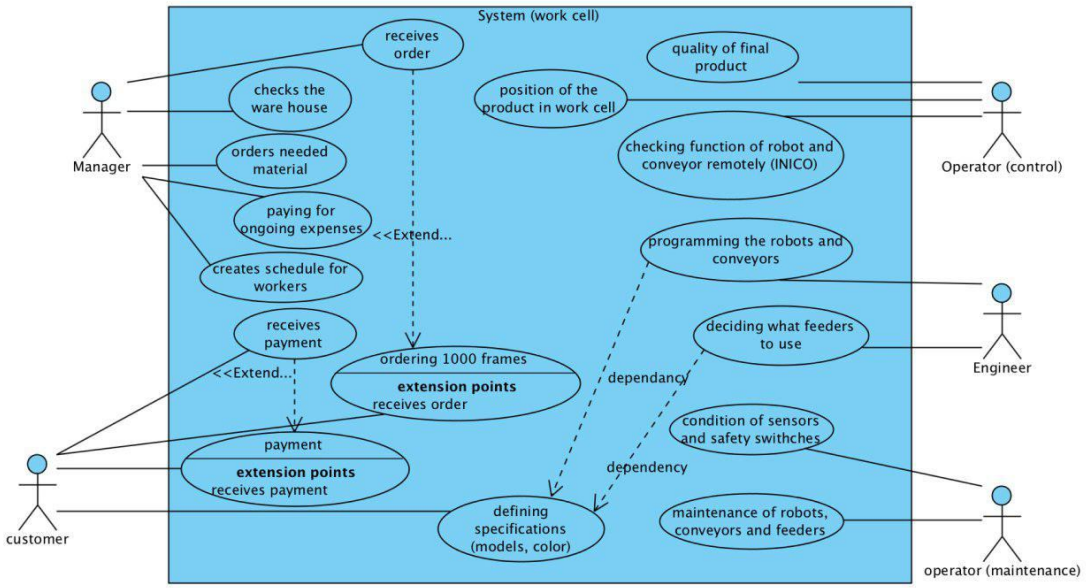
Use Case Diagram

Our Use case diagram has improved a lot through iterations as our understanding of use cases and actors in the monitoring system has changed. Follows below our use case diagrams as consecutively has been improved:

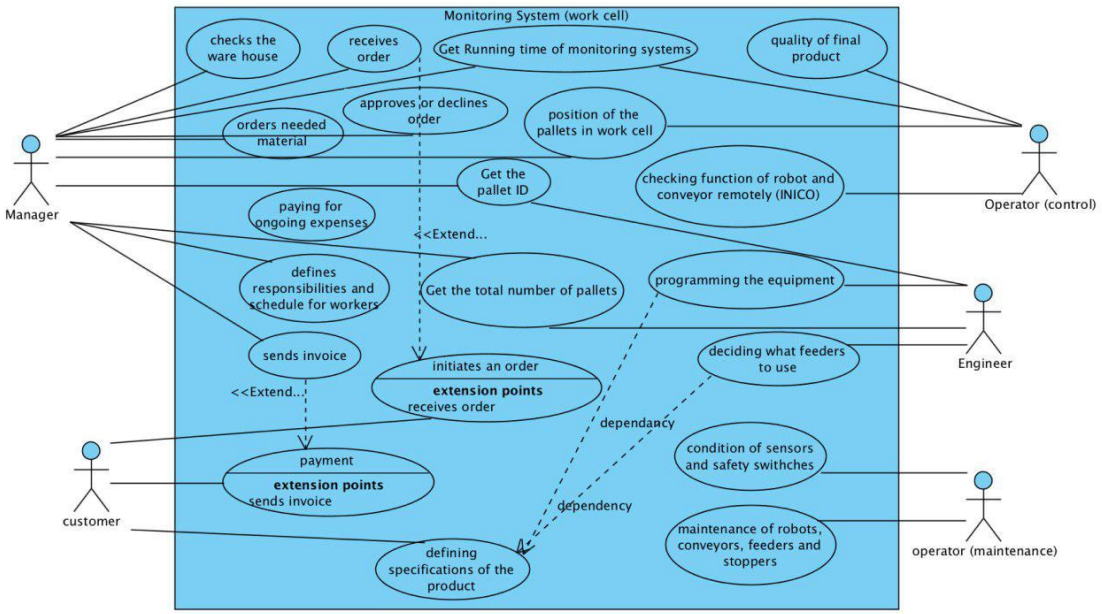
First iteration:



Second iteration:



Final iteration:

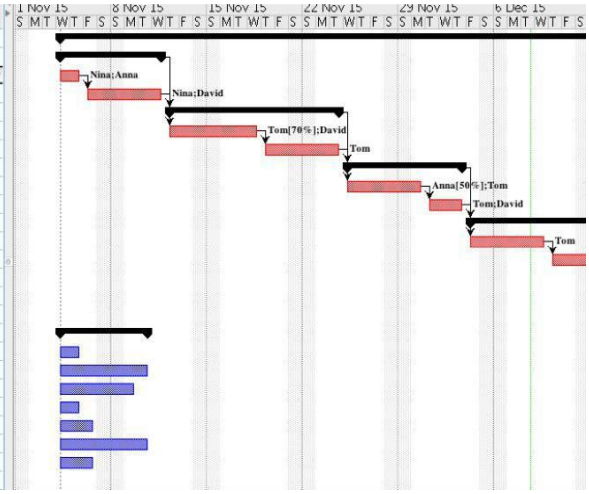


WBS

Our WBS has also improved according to our use case diagram. Below the diagrams are shown:

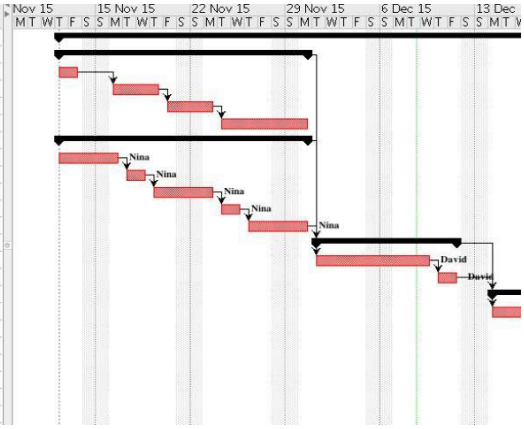
First iteration:

	Name	Duration	Cost	Start
1	⊟inception iteration	29 days	$10440.00	11/4/15 8:00 AM
2	⊟Business Model	6 days	$2560.00	11/4/15 8:00 AM
3	Define the Business Environment	2 days	$800.00	11/4/15 8:00 AM
4	Define System and Project Scope	4 days	$1760.00	11/6/15 8:00 AM
5	⊟logistic and planing	9 days	$3200.00	11/12/15 8:00 AM
6	project manager(s)	5 days	$2240.00	11/12/15 8:00 AM
7	stakeholders	4 days	$960.00	11/19/15 8:00 AM
8	⊟resources	7 days	$3000.00	11/25/15 8:00 AM
9	operator (maintainance)	4 days	$1440.00	11/25/15 8:00 AM
10	operator (quality controll)	3 days	$1560.00	12/1/15 8:00 AM
11	⊟Requirement	7 days	$1680.00	12/4/15 8:00 AM
12	define the work cell's capabilities	4 days	$960.00	12/4/15 8:00 AM
13	functional	3 days	$720.00	12/10/15 8:00 AM
14	⊟Risk & Feasibility	7 days	$1120.00	12/15/15 8:00 AM
15	Develop Cost / Benefit	4 days	$640.00	12/15/15 8:00 AM
16	Identify Risk	3 days	$480.00	12/21/15 8:00 AM
17	⊟Project Environment	5 days	$0.00	11/4/15 8:00 AM
18	safety switches	2 days	$0.00	11/4/15 8:00 AM
19	SCARA robot	5 days	$0.00	11/4/15 8:00 AM
20	conveyor	4 days	$0.00	11/4/15 8:00 AM
21	feeders	2 days	$0.00	11/4/15 8:00 AM
22	web browser	3 days	$0.00	11/4/15 8:00 AM
23	product developement	5 days	$0.00	11/4/15 8:00 AM
24	controlling devices (INICO)	3 days	$0.00	11/4/15 8:00 AM

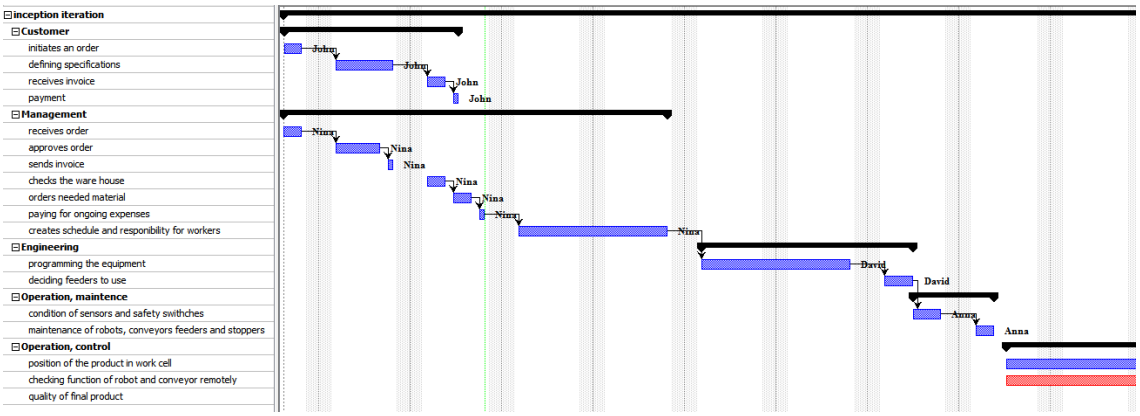


Second iteration:

	Name	Duration	Start	Finish
1	**inception iteration**	39 days	**11/12/15 8:00 AM**	**1/5/16 5:00 PM**
2	**Customer**	13 days	**11/12/15 8:00 AM**	**11/30/15 5:00 PM**
3	receives payment	2 days	11/12/15 8:00 AM	11/13/15 5:00 PM
4	ordering 1000 frames	4 days	11/16/15 8:00 AM	11/19/15 5:00 PM
5	payment	2 days	11/20/15 8:00 AM	11/23/15 5:00 PM
6	defining specifications	5 days	11/24/15 8:00 AM	11/30/15 5:00 PM
7	**Management**	13 days	**11/12/15 8:00 AM**	**11/30/15 5:00 PM**
8	receives order	3 days	11/12/15 8:00 AM	11/16/15 5:00 PM
9	checks the ware house	2 days	11/17/15 8:00 AM	11/18/15 5:00 PM
10	orders needed material	3 days	11/19/15 8:00 AM	11/23/15 5:00 PM
11	paying for ongoing expenses	2 days	11/24/15 8:00 AM	11/25/15 5:00 PM
12	creates schedule for workers	3 days	11/26/15 8:00 AM	11/30/15 5:00 PM
13	**Engineering**	9 days	**12/1/15 8:00 AM**	**12/11/15 5:00 PM**
14	programming the robots and conveyors	7 days	12/1/15 8:00 AM	12/9/15 5:00 PM
15	deciding feeders using	2 days	12/10/15 8:00 AM	12/11/15 5:00 PM
16	**Operation, maintence**	6 days	**12/14/15 8:00 AM**	**12/21/15 5:00 PM**
17	condition of sensors and safety swithches	3 days	12/14/15 8:00 AM	12/16/15 5:00 PM
18	maintenance of robots, conveyors and feeder	3 days	12/17/15 8:00 AM	12/21/15 5:00 PM
19	**Operation, control**	11 days	**12/22/15 8:00 AM**	**1/5/16 5:00 PM**
20	quality of final product	3 days	12/22/15 8:00 AM	12/24/15 5:00 PM
21	position of the product in work cell	4 days	12/25/15 8:00 AM	12/30/15 5:00 PM
22	checking function of robot and conveyor remo	4 days	12/31/15 8:00 AM	1/5/16 5:00 PM



Fourth iteration:



Cost benefit analysis

Our cost and benefit analysis has not have any changes since it is all about considering imaginary case of expenditure and benefit.

		Current Year (CY)	CY +1	CY +2	CY +3	CY +4	CY +5		
Costs									**Cost Benefit Analysis**
1	Purchase	$ 200 000,00	$ 200,00	$ 100,00	$ 50,00	$ 50,00	$ 50,00		Total PV Benefits $ 1 156 873,57
2	Labor	$ 20 000,00	$ 22 000,00	$ 23 000,00	$ 23 500,00	$ 24 000,00	$ 25 000,00		Total PV Costs $ 381 205,10
3	Maintenance	$ 1 000,00	$ 1 000,00	$ 1 050,00	$ 1 100,00	$ 1 150,00	$ 1 200,00		**NET BENEFIT** **775 668,48**
4	Energy	$ 5 000,00	$ 5 000,00	$ 5 000,00	$ 5 000,00	$ 5 000,00	$ 5 000,00		
5	Software	$ 10 000,00	$ 1 500,00	$ 1 200,00	$ 1 000,00	$ 1 000,00	$ 1 000,00		
	Total Costs (Future Value)	$ 236 000,00	$ 29 700,00	$ 30 350,00	$ 30 650,00	$ 31 200,00	$ 32 250,00		
	Total Costs (Present Value)	**$ 236 000,00**	**$ 29 117,65**	**$ 29 171,47**	**$ 28 882,18**	**$ 28 823,98**	**$ 29 209,82**	$ 381 205,10	
Benefits									
1	Sales of Nokia Phones X10	$ 20 000,00	$ 40 000,00	$ 75 000,00	$ 80 000,00	$ 100 000,00	$ 150 000,00		
2	Sales of Nokia Phones X100	$ 23 000,00	$ 20 000,00	$ 32 000,00	$ 23 100,00	$ 222 000,00	$ 32 320,00		
3	Sales of Nokia Phones X1000	$ 40 000,00	$ 50 000,00	$ 65 200,00	$ 76 000,00	$ 87 000,00	$ 98 000,00		
	Total Benefits (Future Value)	$ 83 000,00	$ 110 000,00	$ 172 200,00	$ 179 100,00	$ 409 000,00	$ 280 320,00		
	Total Benefits (Present Value)	**$ 83 000,00**	**$ 107 843,14**	**$ 165 513,26**	**$ 168 769,93**	**$ 377 852,78**	**$ 253 894,46**	$ 1 156 873,57	
	Present Value Discout Rate	**2 %**							

CRUD analysis

CRUD analysis template has been modified and improved in this way:

First iteration:

CRUD Matrix (C=Create, R=Read, U=Update, D=Delete)	Manager	Engineer	Operator (Control)	Operator (Maintenance)	Customer	Product	Robot	Conveyor	Feeder
Inspecting quality of final product	R	R	CRU			R			
Checking position of the product in work cell	R	R	CRU			R	R	R	R
Checking function of robot and conveyor remotely (INICO)	R	R	CRU				R		R
Programming the robots and conveyors	R	CRU					R	R	
Deciding what feeders to use	R	R	CRU						R
Monitoring condition of sensors and safety swithches	R	R	CRU						
Maintenance of robots, conveyors and feeders	R	R		RU		R		R	R
Defining specifications (models, color)	R	CRU							
Receives order	CRUD								
Checks the ware house	R								
Orders needed material	CRUD								
Paying for ongoing expenses	CRUD								
Creates schedule for workers	CRUD								
Payment	R				CRU				

Second iteration:

	CRUD Matrix (C=Create, R=Read, U=Update, D=Delete)	Manager	Engineer	Operator (Control)	Operator (Maintenance)	Customer	Product	Robot	Conveyor	Feeder
2	Inspecting quality of final product	R	R	CRU			R			
3	Checking position of the product in work cell	R	CRUD	CRU			R	R	R	R
4	Checking function of robot and conveyor remotely (INICO)	R	R	CRU				R		R
5	Programming the robots and conveyors	R	CRU					R	R	
6	Deciding what feeders to use	R	R	CRU						R
7	Monitoring condition of sensors and safety swithches	R	R	CRU						
8	Maintenance of robots, conveyors and feeders	R	R		RU		R		R	R
9	Defining specifications (models, color)	R	CRU							
10	Receives order	CRUD								
11	Checks the ware house	R								
12	Orders needed material	CRUD								
13	Paying for ongoing expenses	CRUD								
14	Creates schedule for workers	CRUD								
15	Payment	R				CRU				
16	Get the ID of pallets							R		
17	Get the running time of system	CRUD	RU	RU	RU					
18	Get the total number of pallets	CRUD	RU	RU	RU			R		
19	Get the loacation of Pallets	CRUD	RU	RU	RU					
20	Control the movement of converyors	CRUD	RU	RU	RU				RU	
21	Control and test the robot	CRUD	RU	RU	RU			RU		

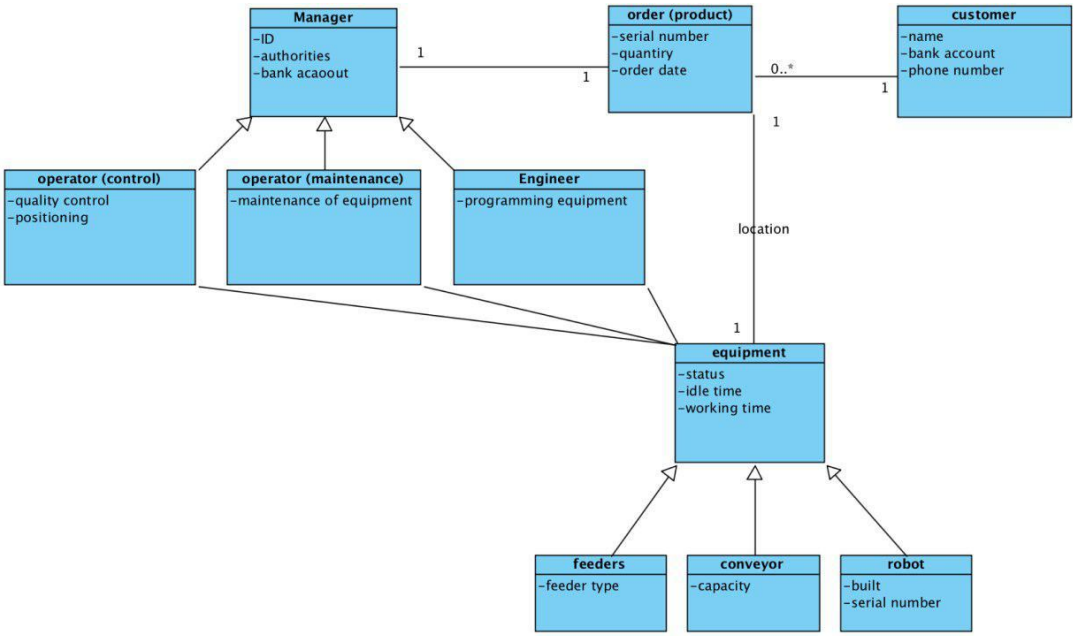
Fourth iteration:

CRUD Matrix (C=Create, R=Read, U=Update, D=Delete)	Manager	Engineer	Operator (Control)	Operator (Maintenance)	Customer	Product	Robot	Conveyor	Feeder
Inspecting quality of final product	R	R	CRU			R			
Checking position of the product in work cell	R	R	CRU			R	R	R	R
Checking function of robot and conveyor remotely (INICO)	R	R	CRU				R		R
Programming the robots and conveyors	R	CRU					R	R	
Deciding what feeders to use	R	R	CRU						R
Monitoring condition of sensors and safety swithches	R	R	CRU						
Maintenance of robots, conveyors and feeders	R	R		RU		R		R	R
Defining specifications (models, color)	R	CRU							
Receives order	CRUD								
Checks the ware house	R								
Orders needed material	CRUD								
Paying for ongoing expenses	CRUD								
Creates schedule for workers	CRUD								
Payment	R				CRU				

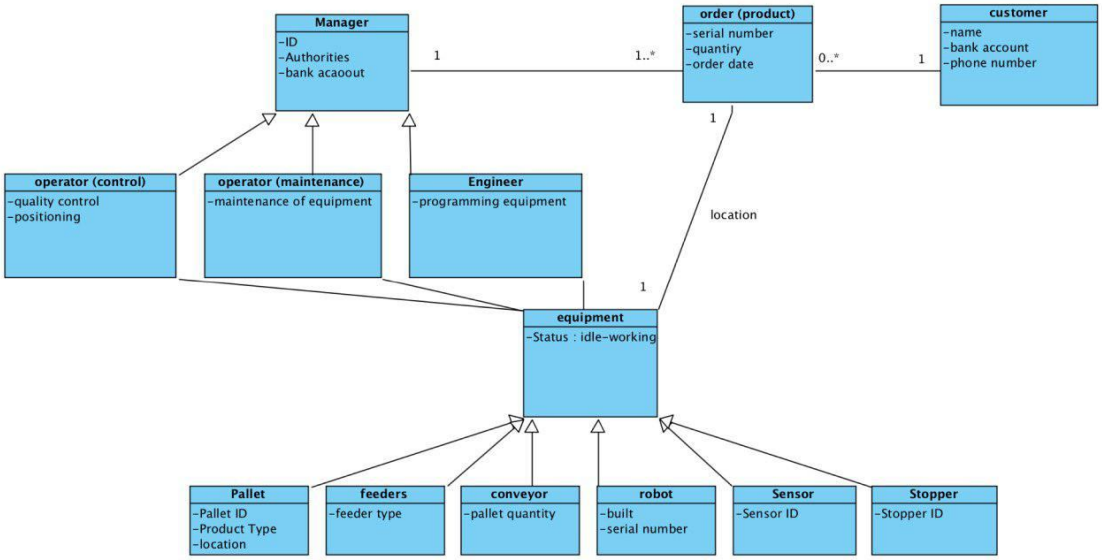
Class diagram

For the class diagram also our understanding of classes needed in monitoring system has improved. Our class diagrams from second until last iteration is presented:

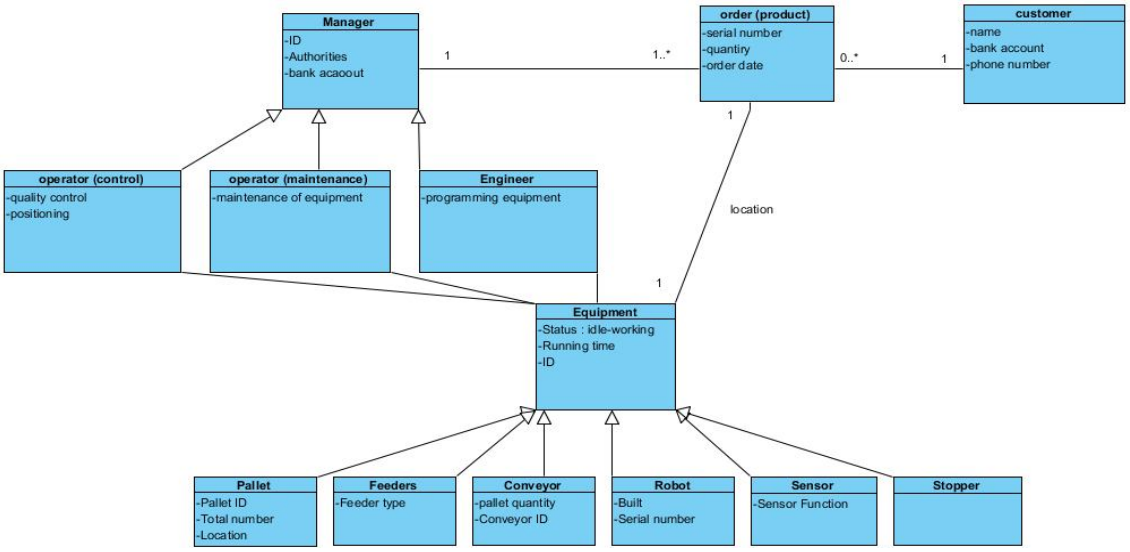
second iteration



Third iteration:



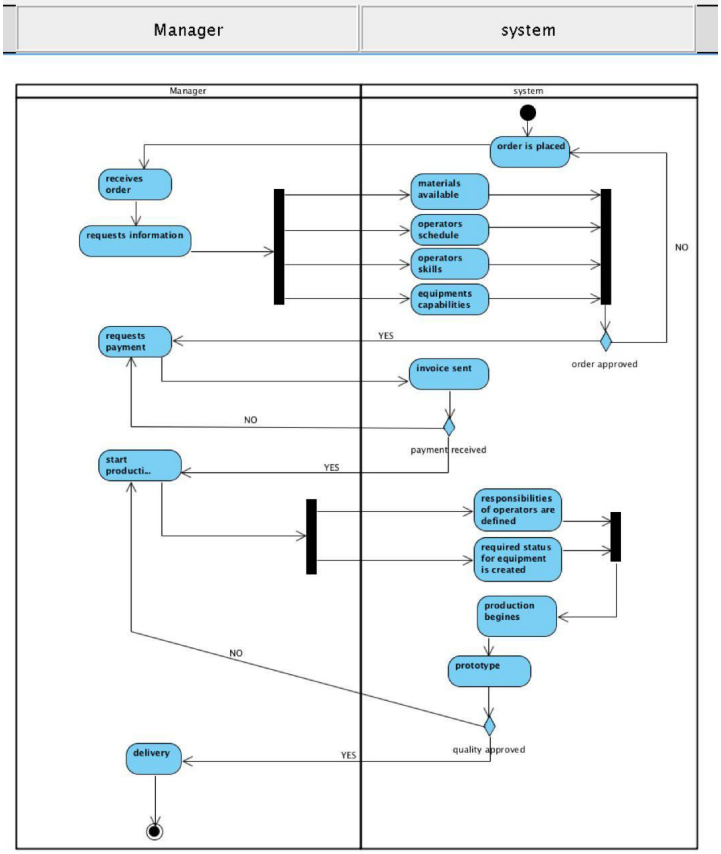
Final iteration:



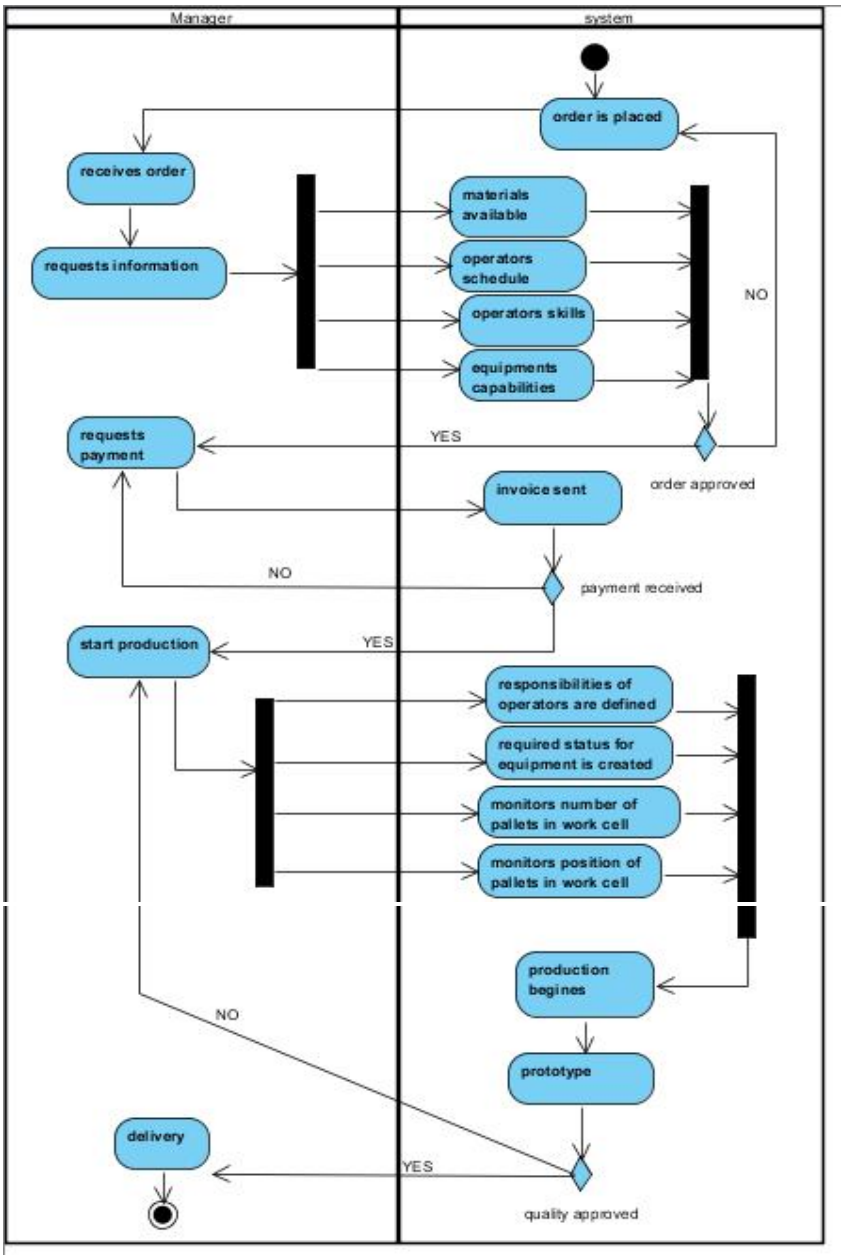
Activity diagram

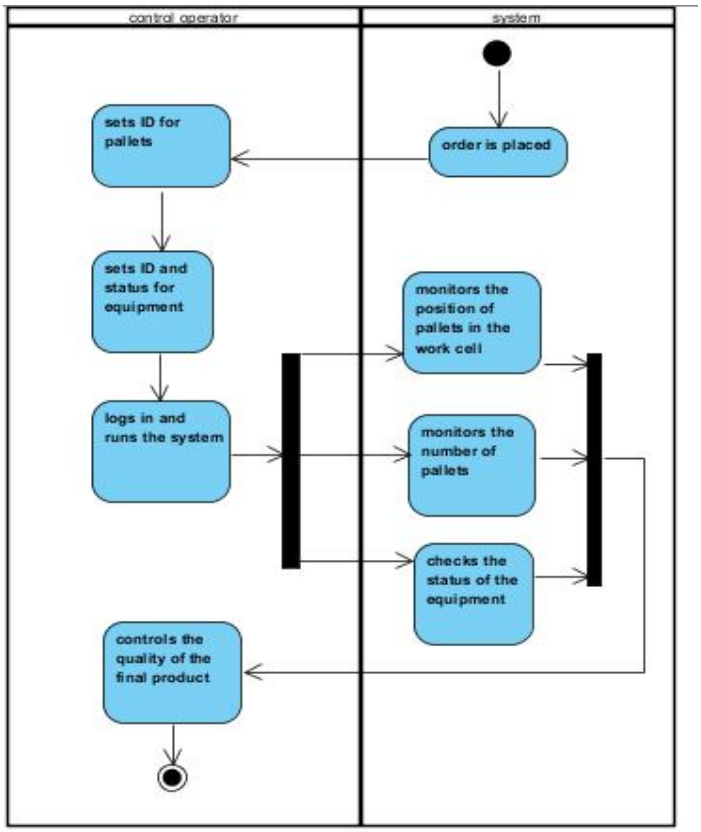
Our activity diagram was first created in 3rd iterations and after that other actors and activities was added to it and long with some modifications was presented in fourth and final iteration:

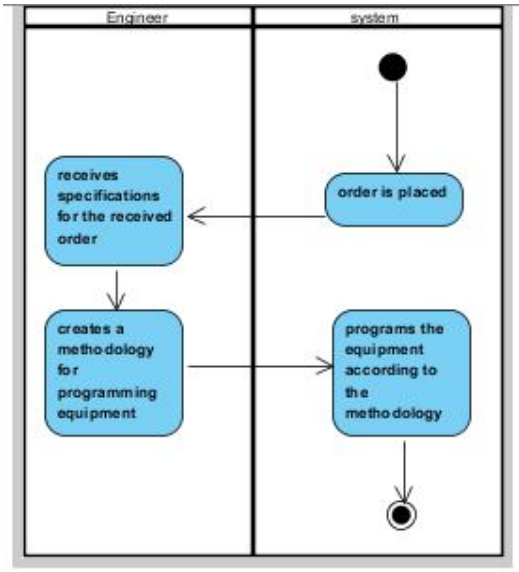
Third iteration:

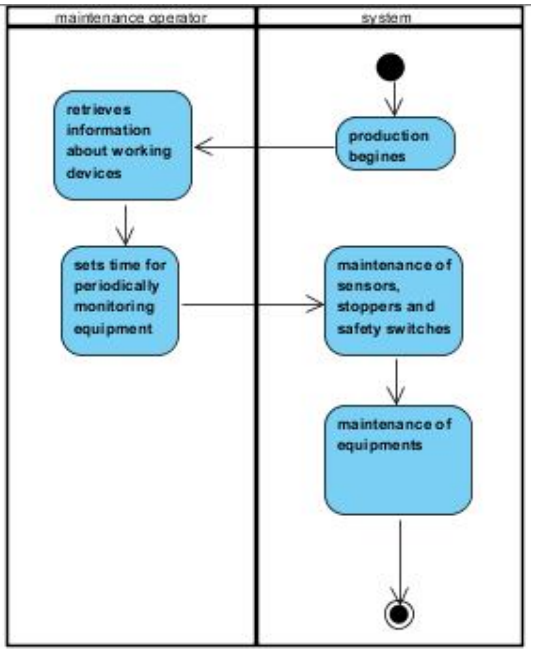


Fourth and final iteration:




control operator
system
sets ID for pallets
order is placed
sets ID and status for equipment
monitors the position of pallets in the work cell
logs in and runs the system
monitors the number of pallets
checks the status of the equipment
controls the quality of the final product

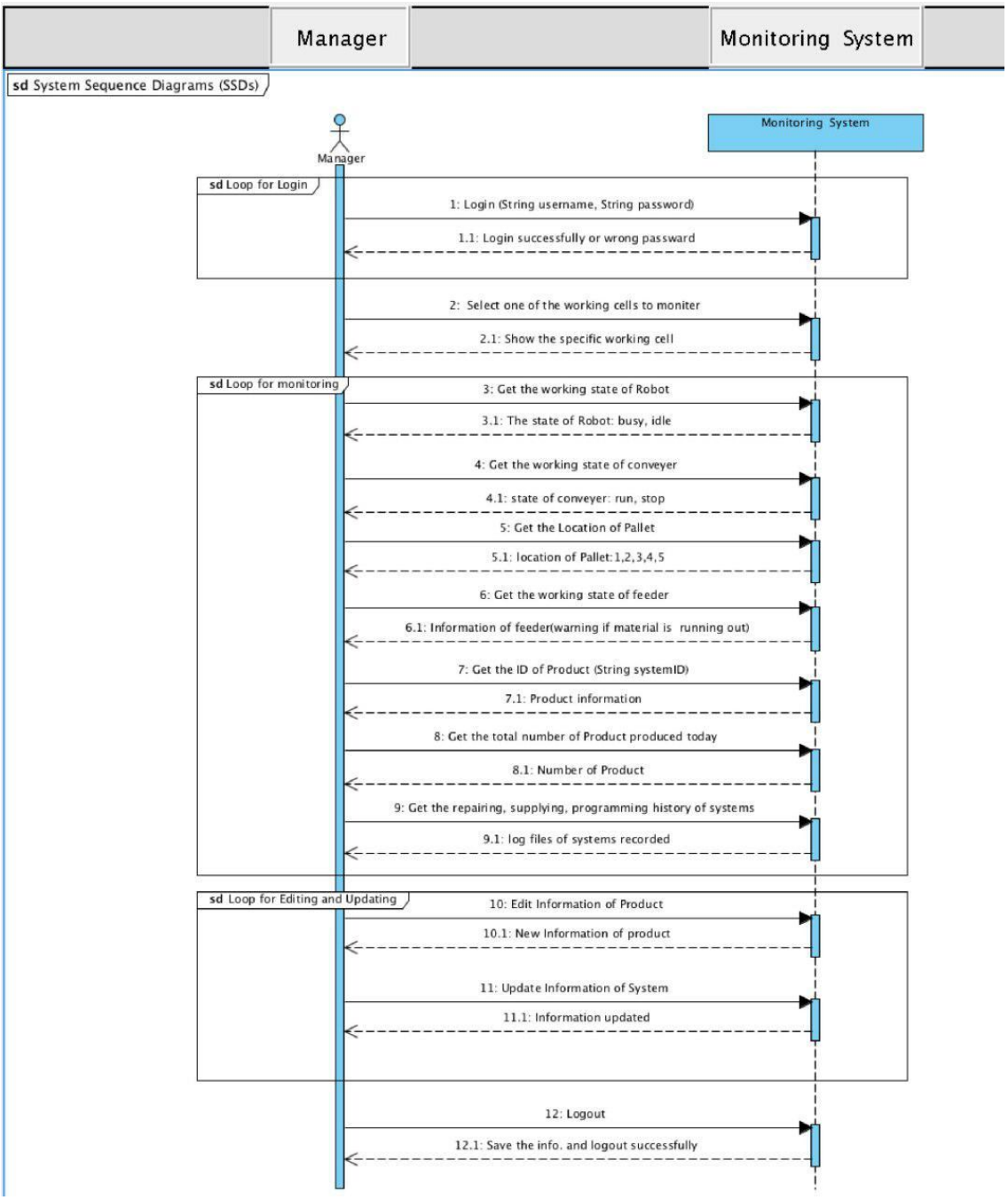

Engineer
system
receives specifications for the received order
order is placed
creates a methodology for programming equipment
programs the equipment according to the methodology


maintenance operator
system
retrieves information about working devices
production begines
sets time for periodically monitoring equipment
maintenance of sensors, stoppers and safety switches
maintenance of equipments

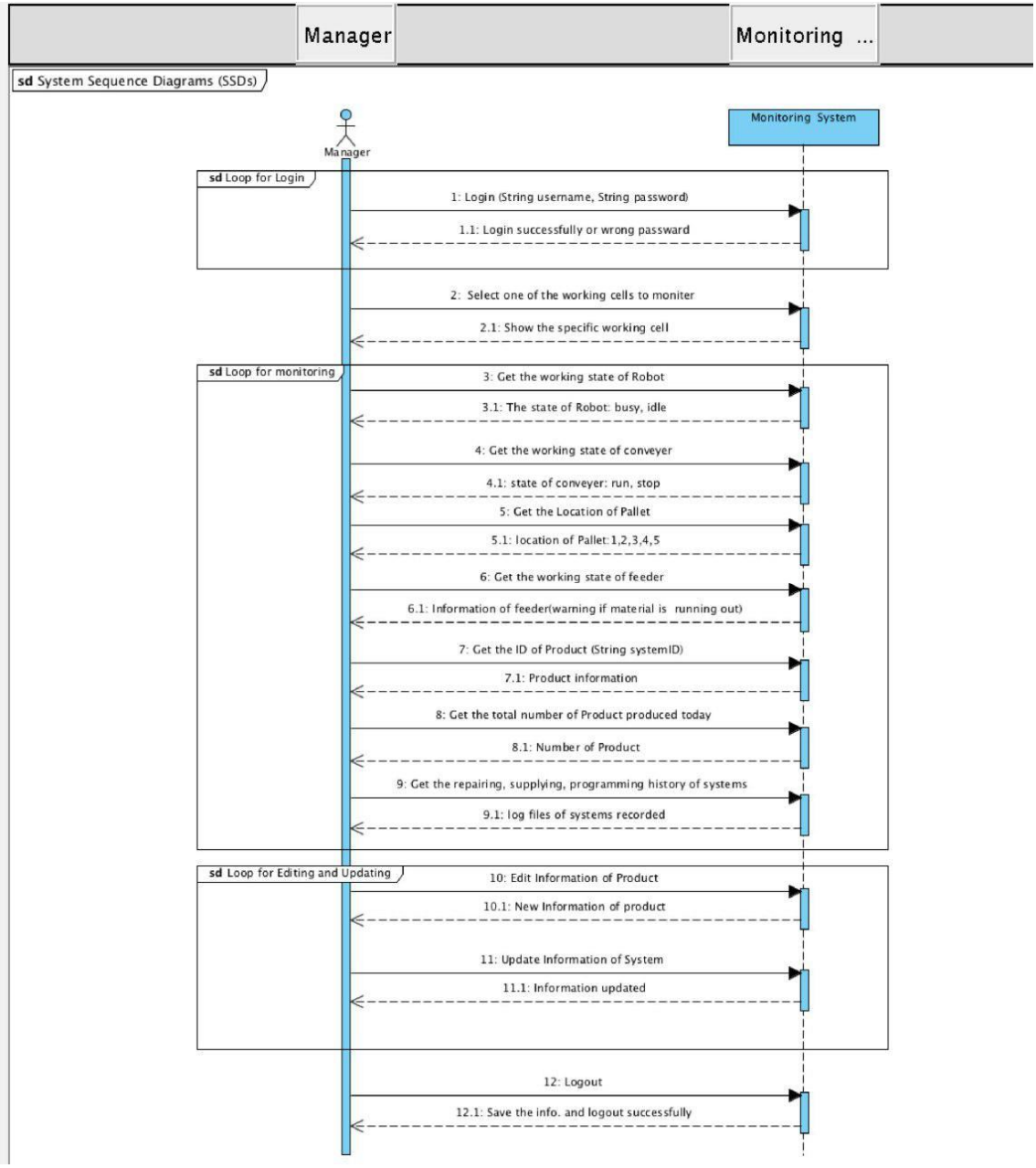
Sequence diagram

Our sequence diagram has also been improved the same way form third through final iteration:

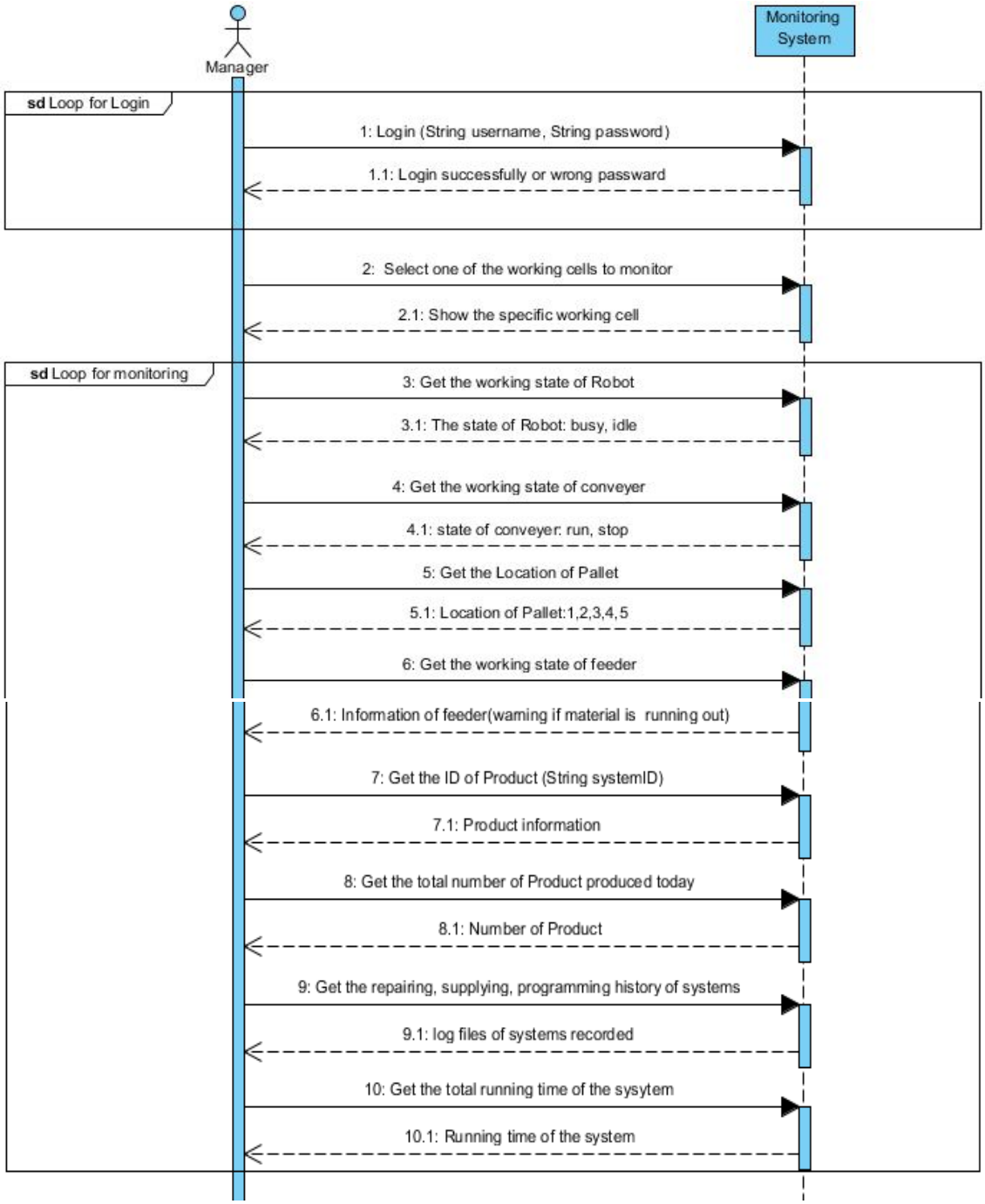
Third iteration:

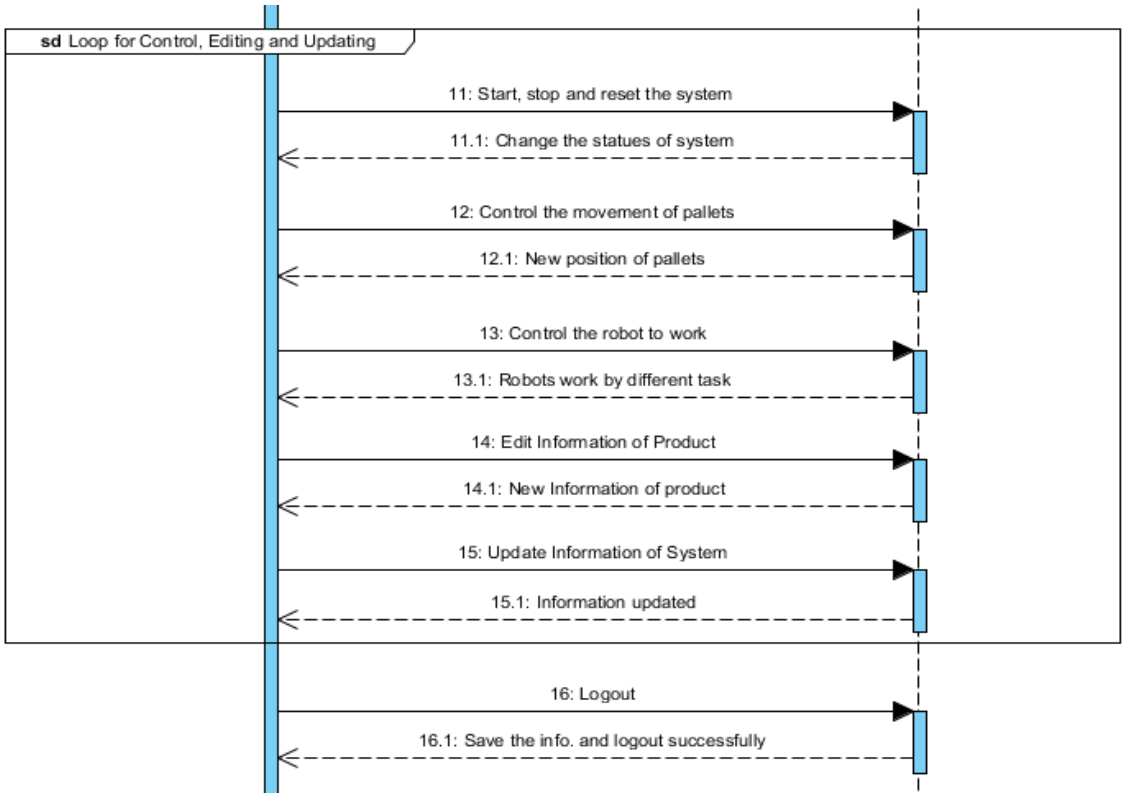


Fourth iteration:



Final iteration:

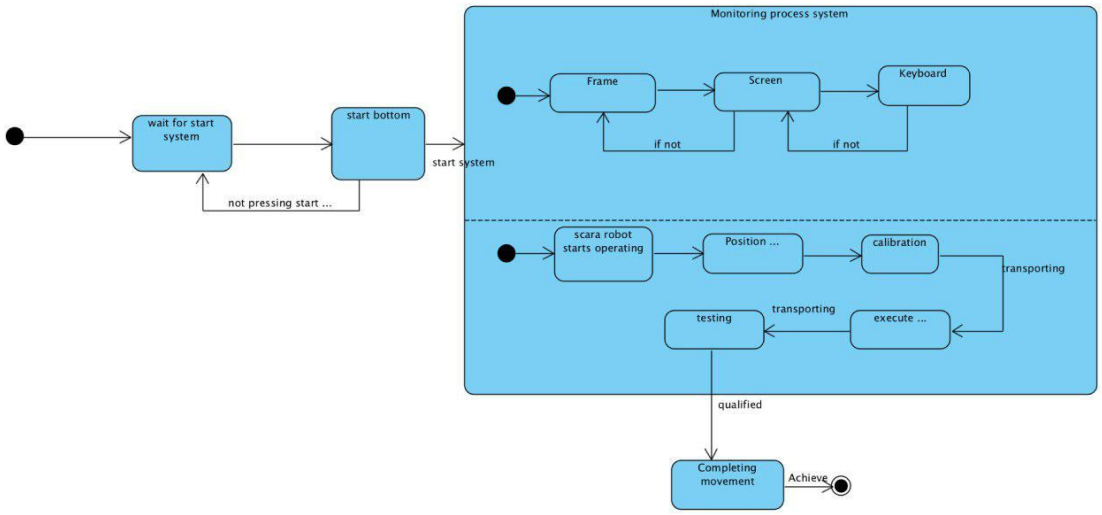




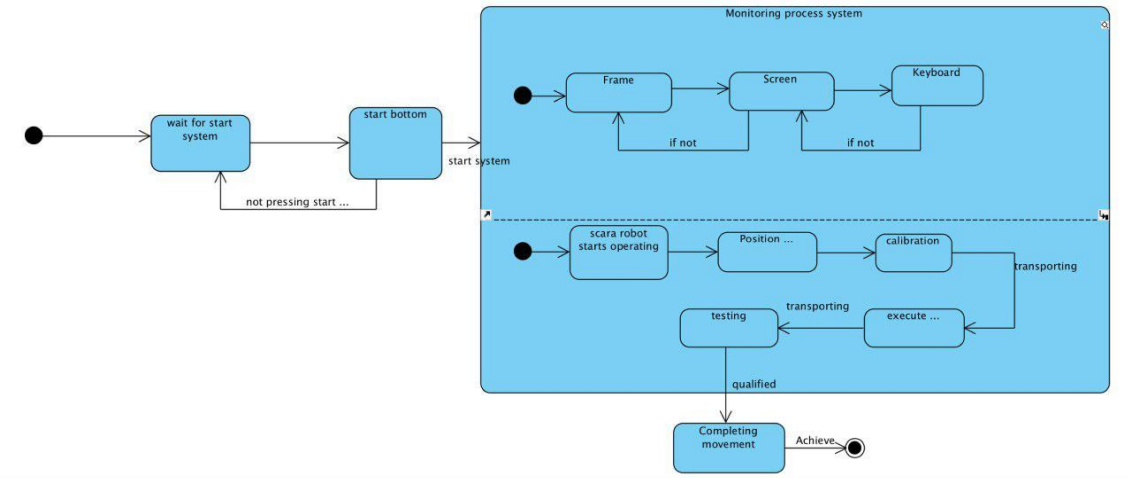
State diagram

Our state diagram has improved a lot through iterations:

Third iteration:



Fourth iteration:



Final iteration: